Load Management for Publish/Subscribe Message Oriented Middleware

Ran TAO

Submitted for the degree of Doctor of Philosophy

School of Electronic Engineering and Computer Science

Queen Mary, University of London

June 2015

To my beloved family

ACKNOWLEDGEMENT

I would like to express my deepest and sincerest gratitude to my supervisor, Dr. Stefan Poslad for his consistent support, patient guidance, and professional supervision. His wide knowledge and enthusiasm towards research have been of great value to me.

My heartfelt thanks go out to Dr. John Bigham for his guidance and support that have given to me. His valuable ideas and helpful advice have helped immensely in my PhD research. My sincere appreciation also goes to Dr. Felix Cuadrado, Dr. Laurissa Tokarchuk, and Dr. Gareth Tyson for their help and advice.

I am extremely grateful to all my friends in the Network Research Group at QMUL, for their sincere friendship, supportive encouragement and valuable suggestions.

Finally, I would like to thank my beloved family; my parents who raised me, who continue to love and support me over many years, and my dear wife, Shiwen Zhou, whose patience and faithful support is so appreciated.

ABSTRACT

To provide time-critical early warnings in a Tsunami Warning System (TWS), the time delay for both the sensor data exchange process (upstream) and the warning message dissemination processes (downstream) should be minimal, maximising the time available for accurately analysing the situation and giving more time for people in the affected region to react to the warnings. Publish/Subscribe Message-oriented Middleware (PSMOM) in combination with a novel use of a federated broker (broker overlay) can be deployed in TWS to support both time-critical and resilient communication. PSMOM can better manage message bursts caused by a sudden increase in sensor data exchange frequency or by additional sensors coming online. PSMOM can better manage the decrease in available system resources (bottlenecks) caused by a disruption in the underlying network infrastructure (limited resource case). Otherwise, these burst and bottlenecks can cause some brokers to become overloaded, which may in turn degrade the overall system performance and delay decision-making.

Existing PSMOM load management solutions have two key limitations when applied to TWS. First, existing work does not consider the message delay requirements for the redistribution and offloading phases of load management. Here, some data is only useful or valid for a short time-span (from tens of seconds to tens of minutes); hence, it needs to be exchanged within this maximum allowed end-to-end transmission delay. Time critical subscribers need to be de-prioritised from being offloaded, as the offloading processes take some time to complete, introducing unexpected delays to message exchange. Second, existing solutions assume that there are surplus system resources for offloading, i.e., less loaded brokers can accept loads from overloaded brokers. However, in a TWS, the underlying network infrastructure may be disrupted which in turn reduces the system capacity. It always takes time to recover from the limited resources situation and during that time, brokers may not have enough system resources to accept loads from overloaded brokers, which may result in total failure of the overloaded brokers.

A novel load management framework called ePEER is proposed that extends an existing messaging system, Publish/Subscribe Efficient Event Routing (PEER), with the following main contributions. First, for the surplus resource case, the message delay

requirements for different subscription services are considered in the load analysis process when offloading the load to different brokers. Second, for the limited resource case, a feedback driven congestion control mechanism can be used when the underlay network infrastructure is damaged, reducing the available bandwidth of PSMOM. This mechanism limits the publication rate of messages with less value, to better maintain the quality of experience (QoE) of subscribers for the more important messages

ePEER is validated with emulation-based experiments. The results show that ePEER outperforms the state of the art load management solution used by PEER: through preventing unnecessary delays introduced to time critical services, and through ensuring important messages can be more efficiently exchanged to improve the QoE of subscribers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ADT	Advertisement Table
ADV	Advertisement
AMQP	Advanced Message Queuing Protocol
CAP	Common Altering Protocol
CTWP	Candidate Tsunami Watch Provider
DRD-LB	Delay Requirement Driven Load Balancing
EMA	Edge Management Agent
ePEER	Extended Publish/Subscribe Efficient Event Routing
EWS	Early Warning System
FDCC	Feedback-Driven Congestion Control
FIFO	First In First Out
H-E	Head-Edge
HMA	Head Management Agent
IB	Intermediate Broker
JMS	Java Messaging Service
LA	Load Analyser
LAN	Local Area Network
LB	Load Balancer
LD	Load Detector
LM	Load Manager
MA	Management Agent
MOM	Message Oriented Middleware
MQTT	Message Queuing Telemetry Transport
NEAMTIC	North-Eastern Atlantic and Mediterranean Tsunami Information Centre
NTWC	National Tsunami Warning Centre
OGC O&M	Open Geospatial Consortium Observations & Measurements
OM	Overlay Manager
OutBW	Output Bandwidth
P2P	Peer to Peer
PDCC	PHB Driven Congestion Control
PE	Pubend (Publisher Endpoint)
PEER	Publish/subscribe Efficient Event Routing

PHB	Publisher Hosting Broker
PS	Publish/Subscribe
PSMOM	Publish/Subscribe Message Oriented Middleware
PTL	Publication Topic List
QoE	Quality of Experience
RTWC	Regional Tsunami Watch Centre
SA	Speed Analyser
SDCC	SHB Driven Congestion Control
SHB	Subscriber Hosting Broker
SLA	Server Level Agreement
STOMP	Streaming Text Orientated Message Protocol
TNC	Tsunami National Contact
TS	Topic Selector
TWFP	Tsunami Warning Focal Point
TWS	Tsunami Warning System
UA	Utility Analyser
WAN	Wide Area Network

1 INTRODUCTION

1.1 Application Driven Motivation

Natural environment disasters, caused by natural events such as tsunamis, or manmade crises such as earth substrate drilling, cause widespread environment damage that may take the affected regions years to recover after the onset of the disaster. An Early Warning System or EWS is a core system used for environment disaster risk and effect reduction. It helps prevent loss of lives and reduces the economic and material impacts of disasters [1]. To be effective, a EWS needs to actively involve the communities at risk, facilitate public education and awareness of risks, effectively disseminate messages and warnings and ensure there is constant state of preparedness [1]. A functional EWS can be implemented as a chain of information communication systems. It comprises sensors, event detection, decision support, and message broker subsystems in a given order. It can be used for forecasting and signalling disturbances that adversely affecting the stability of that part of the physical world being monitored. It helps to give sufficient time for the response system to prepare resources and response actions to minimise the impact on the stability of the physical world [2]. A Tsunami Warning System (TWS) is chosen as the motivating application of a EWS.

In a TWS, a tsunami is detected through the analysis of seismic and oceanographic data gathered by physical sensors, e.g., seismometers, tide gauges, and coastal buoys. The sensor data is gathered and transmitted (upstream) to remote off-site tsunami operation centres, which run the routine operation event and special event detection processes and generate addition data flows (downstream) to enable decision handling processes and the command-control workflows. Tsunami operation centres are often located in different regions (or countries). They work collaboratively to detect tsunami events, working with government offices to disseminate tsunami warnings. Take the North-Eastern Atlantic and Mediterranean Tsunami Information Centre (NEAMTIC) for example, the operation centres, named Candidate Tsunami Watch Providers (CTWPs) or Regional Tsunami Warning Centres (RTWCs), are geo distributed in different countries such as Portugal, Italy, Greece, France, and Turkey. These centres collect, record, and process earthquake data for the initial warning and further collect, record and process the sea-level data for confirming or cancelling the initial warning. The

warnings are sent to the Tsunami Warning Focal Points (TWFPs) or National Tsunami Warning Centres (NTWCs), which are available at the national level for issuing tsunami event information [3]. Figure 1-1 shows the geo-distribution of Tsunami National Contacts (TNCs)[1], TWFPs and CTWPs in the NEAM region [4].

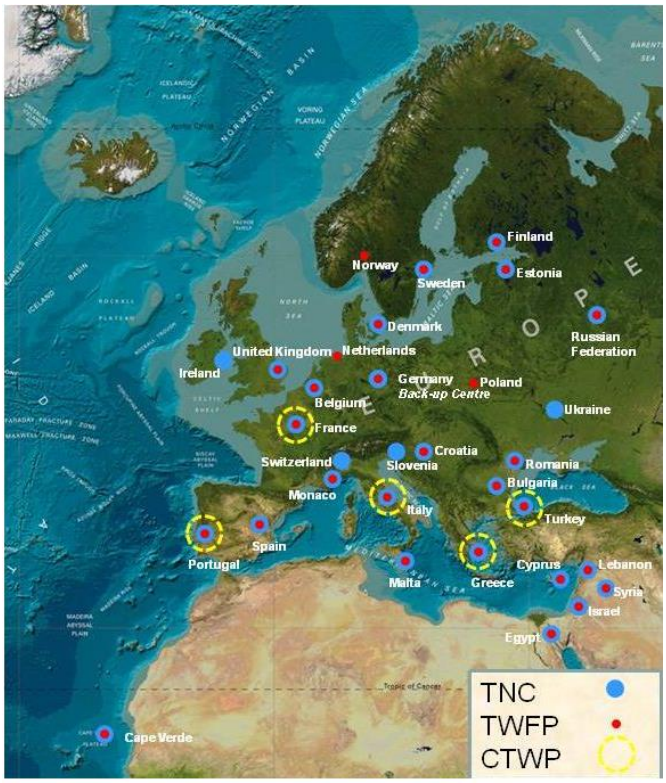

Figure 1-1 Established TNCs, TWFPs and CTWPs in the NEAM Region

In this thesis, the focus of the research is on a TWS's communication system. This enables the (upstream) data exchange from physical environment sensors that publish their environment data to the corresponding operation centres; and enables the (downstream) messages exchange for warning and alert message dissemination. The downstream system also includes data storage and analysis processes, e.g., resilient database design, risk assessment, and algorithms to predict the crisis, but these are considered out of the scope of the thesis. Upstream and downstream communications have three main requirements for time criticality, scalability and resilience.

[1] TNC is a person who represents his/her country in the coordination of international tsunami warning and mitigation activities.

1) Time Criticality

For an effective tsunami warning, time is of the essence [5]. The sum of the detection time (T1), assessment time (T2), and evacuation time (T3) must be less than the tsunami travel time (T4), i.e., T1+T2+T3 < T4. The time needed for a tsunami warning time is illustrated in Figure 1-2.

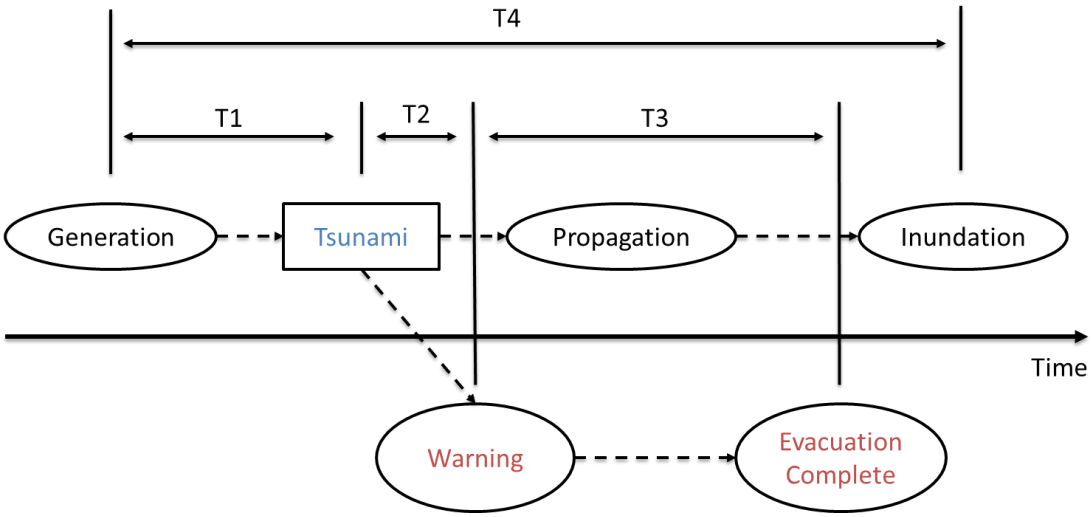

Figure 1-2 Time Needed for a Tsunami Warning

The tsunami travel time T4 depends upon the distance between the sea or earthquake epicentre (start point) and the shore, e.g., if the distance is less than 400 km, it takes less than about 30 minutes for a tsunami to reach the shore [6]. In 2004, the tsunami in Indonesia hit Aceh, a coastal city, within 15 minutes. In these cases of near field tsunamis, there is little lead time for a tsunami warning [6]. Therefore, the upstream and downstream message exchange times to issue a warning should be minimised.

2) Scalability and Resilience

If a region's tsunami sensors indicate that a tsunami is likely to occur, e.g., because of increased movement by tsunami buoys, the sensor data generation rate increases because sensors' sampling rates in a TWS are designed to increase automatically. This may cause an information flood. Thus, both the upstream and downstream communication need to be scalable, i.e., to be scaled-up using more system resources such as memory and bandwidth, and scaled-down when fewer resources are required, or are available. In addition, when a tsunami reaches land, the embedded upstream network infrastructure can be disrupted in the affected area, affecting the

communication capacity and availability. To ensure the upstream sensor data reaches receivers in the operation centres, the upstream communication needs to be designed to be resilient, e.g., using guaranteed delivery, mirroring, overlay routing, and congestion control. Note that it is presumed that the downstream communication to the operation centre is remote to, and away from, the region of the environment disaster and thus the data processing centre is less prone to be disrupted.

Several communication systems can support the upstream and downstream data communication in a TWS, e.g., Remote Procedure Call (RPC), brokerless system such as ZeroMQ[2], and Publish/Subscribe Message Oriented Middleware (PSMOM). Among these systems, PSMOM is selected as the focus in the thesis. The details of the comparison among these systems are provided in Section 2.2.4. In a PSMOM, publishers are clients that generate and send messages to an intermediary called a message broker; subscribers are clients that receive and may process messages from a broker [7]. A PSMOM supports both synchronous and asynchronous message exchange, which is advantageous when publishers and subscribers are temporally and spatially distributed [7, 8]. A PSMOM also supports one-to-many, many-to-one and many-to-many communications [9-11]. This is an efficient way to exchange messages. The same message may need to be published to multiple subscribers, e.g., a warning message is disseminated to multiple receivers. A subscriber can combine information from multiple publishers, e.g., a database receives both raw and processed data. Multiple publishers may publish messages to multiple subscribers, e.g., multiple operation centres may process data. PSMOM enables subscribers to select which messages they receive. These functions above are based upon filtering messages that either match their metadata descriptions which are defined as message topics (topic-based PSMOM) or match a set of filter criteria specified for message content (content-based PSMOM).

In PSMOM, a broker decouples publishers in time and space from subscribers. It receives messages from publishers, detects whether or not messages have any matched subscribers, and forwards them to subscribers that have matched interests or can discard them if no matched subscribers are found. To enable distributed message exchange

[2] ZeroMQ: The Intelligent Transport Layer, from: http://www.zeromq.org/

across a wide area, brokers in different geo-locations can be federated, i.e., messages are disseminated amongst multiple brokers. The federated brokers form an overlay network, in which their published messages are always routed from the source broker that hosts the publisher through a federation path, to the destination brokers, to which the matched subscribers subscribe. Both topic-based and content-based PSMOM have been widely employed to provide message exchange services for applications that include stock market monitoring [12], business process execution [13], activity monitoring [14], workflow management [15], Really Simple Syndication (RSS) filtering [16, 17], and network monitoring and management [14]. In this thesis, topic-based PSMOM is selected to provide data communication services for a TWS, as it provides better performance than content-based PSMOM in this scenario. A comparison between the two types of PSMOM is given in Section 2.2.1. In the remaining parts of the thesis, PSMOM is used to refer to the topic-based PSMOM. According to the communication requirement of a TWS system, a PSMOM needs to provide time-critical message exchange and to be resilient and scalable. Existing work has improved the resilience and scalability [9, 11, 18, 19] of a PSMOM and to lower the transmission delay by using overlay routing [20]. The focus is on the concept of load management for a PSMOM used in TWS in relation to the transmission delay constraints and Quality of Experience (QoE) for subscribers.

In practice, individual brokers in a federated overlay of a PSMOM may become overloaded. This has several potential causes. First, an uneven load distribution may be caused by different capacities of brokers and inter-broker links, and by different population densities, interests, and usage patterns of end-user subscribers [7, 21]. Second, bursts of message exchange may occur when more publishers are added or the publishing rates for some publishers suddenly increases, and thus generates an extra load on brokers. Third, the capacity of the PSMOM system may be reduced due to disruptions of the network infrastructure caused by a physical environment crisis. These broker overload problems may result in performance degradation and delay critical decision-making. Thus, a load management solution is required to manage the broker overlay in TWS. Two cases are considered for the communication load management: surplus ICT resources and limited ICT resources.

1) Surplus Resource Case

In this case, the system has enough communication capacity to provide message exchange services. When some brokers become overloaded, there are surplus resources in other brokers that can accept the load from the overloaded ones. In addition, new brokers can be created for load shifting, e.g., with support for virtual machine management in Cloud Computing.

2) Limited Resource Case

In this case, there are insufficient system resources for all the messages to be exchanged. This is caused by the disruption to the physical network infrastructure. For instance, during the immediate aftermath of the March 2011 Japan Earthquake, at least four major undersea communication cables (two-trans-Pacific and two intra-Asian cables) were damaged [22]. Such damage may severely reduce the capacity of links from publishers to brokers, between federated brokers, and from brokers to subscribers. In this case, offloading through migrating subscribers or publishers from an overloaded broker to another (less loaded) one, does not resolve the broker overload problem. This situation is further exasperated by the exchange of a large amount of messages from overactive publishers, i.e., publishers that have no matched subscribers, whose information may be repetitive or of little value, and where matched subscribers find such content exchange unimportant (Section 5.1). These overactive or non-informative publishers can introduce unnecessary loads for the message exchange, which may overload the computation and communication resources of the communication service (Section 5.2.1).

1.2 Challenges

In this section, the challenges in providing efficient load management for both the surplus resource case and the limited resource case for a PSMOM system are discussed.

1. For the surplus resource case, the key challenge for broker load management is that different TWS message exchange services have different delay requirements.
2. In the limited resource case in TWS, due to the disruption to the underlying network infrastructure, the demand for message exchange by brokers exceeds the supply (of broker resources available). The bandwidth between publishers and

brokers, between federated brokers, and between brokers and subscribers may be used up and cannot be scaled up for a time. New brokers may not be able to be initialised, which means the total broker processing capacity remains limited until the situation recovers. In this case, brokers are temporarily not able to receive and process all messages from the available publishers.

1.3 Research Objectives

The primary research objective is to improve load management for the broker overlay of PSMOM used in TWS, for the surplus resource case, and for the limited resource case. The primary objective is broken down into the following sub-objectives.

1) Analyse the communication requirements in TWS problem domain for different message exchange services.
2) Analyse the limitations of how the state of the art load management solutions can satisfy the TWS communication requirements.
3) Investigate and design a broker overlay that meets the communication requirements for TWS and is capable of enabling resilient information exchange in the face of broker failure and link failure.
4) Investigate and propose load-balancing solutions for a TWS with delay requirements for the surplus resource case.
5) Investigate and propose a feedback driven congestion control solutions for TWS to manage the broker load in the limited resource case.

1.4 Thesis Outline

The remaining of the thesis is organized as follows. Chapter 2 gives an overview of the core concepts needed to understand the analysis of the surveyed methods and the proposed method. Chapter 3 provides a critical analysis of existing load management methods for PSMOM for both the surplus and limited resource cases (literature survey). Chapter 4 describes the delay requirements driven load-balancing solution for the surplus resource case with a simulation-based validation. Chapter 5 proposes a feedback driven congestion control for the limited resource case, which is also validated through simulation-based experiments. Chapter 6 summarizes what has been achieved and proposes some future work.

2 PRELIMINARIES

2.1 Overview

This section, preliminaries, gives an overview of the core concepts needed to understand the analysis of the surveyed methods and the new proposed method. An overview of how PSMOM works and how to deploy PSMOM in TWS is provided in Section 2.2. In Section 2.3, the background of the broker overlay, including the advertising and subscribing process, is described. In Section 2.4, the load management life cycle for PSMOM is described, both for load balancing in the surplus resource case, and for congestion control in the limited resource case.

2.2 Publish/Subscribe Message Oriented Middleware

Message Oriented Middleware or MOM is an infrastructure that focuses on sending and receiving messages and allows message exchange services to be distributed over heterogeneous platforms [23]. It integrates independent, loosely coupled components to increase their interoperability, portability and flexibility, as the participants do not need to know what platforms or processors the others reside on [19, 24-27]. A MOM is typically asynchronous, but most implementations also support synchronous message passing as well.

In a MOM system, there are two typical messaging models, message queuing and publish/subscribe (PS). Message queuing is a peer-to-peer (P2P) communication model where messages are addressed to specific recipients. It is suitable for the request-reply type message exchange. The PS model is a many-to-many model that permits the efficient dissemination of messages across a distributed system [25]. Clients of a PSMOM can be publishers, i.e., information producers that publish messages, or subscribers, i.e., information consumers that subscribe to information of their interest and receive messages [25]. Communication in PSMOM is usually asynchronous [19]; publishers and subscribers are decoupled in time (they do not have to be active at the same time) and space (they do not to be close to each other in the same network). The message exchange does not block the control of flow [28]. Publishers and subscribers do not even need to know of the existence of one other [9, 28]. A significant advantage

of PSMOM is that it reduces the number of point-to-point connections (active communication end-points) in a complex information technology (IT) system [18, 29]. In practice, PSMOM can be implemented in many ways. Java Messaging Service (JMS) is a wide-spread and frequently used middleware technology [30-32]. Several systems are based upon it, such as FioranMQ [33], TibcoEMS [34, 35], WebSphereMQ [36-39], and RabbitMQ [40]. In addition, instead of using a language specific Application Programmer's Interface (API) or Library, such as JMS which only works for Java applications, some standard open network protocol, such as Advanced Message Queuing Protocol (AMQP) [41-43], Message Queuing Telemetry Transport (MQTT) [44] and Streaming Text Orientated Message Protocol (STOMP) [45], can also be adopted to build a PSMOM system.

In PSMOM, messages exchanged consist of a message header and a message body. The message header records protocol metadata, e.g., protocol version. The message body or payload contains the actual data to be exchanged.

Pattern matching is a key characteristic of PSMOMs. It defines the process of matching published messages to subscribers' interests. With respect to matching processes, PSMOM systems can be classified into topic-based and content-based ones [28, 46].

2.2.1 Topic-Based PSMOM & Content-Based PSMOM

In a topic-based PSMOM system, each message is classified as belonging to one of a fixed set of topics, also referred as groups, channels, or subjects. It is the metadata that describes the actual data (stored in the message body) being exchanged, e.g., the weather in London, BBC news. The format of the topic is specified by the message exchange protocol adopted by the PSMOM. For instance, for message exchange service using AMQP, a topic consists of a list of keywords (topic names), separated with a delimiter ".", e.g., "*data.sesnsor.buoy.Turkey*" is a topic that is used when acquiring the sensor data from buoys in Turkey. In practice, topics used in a PSMOM system are pre-defined to meet the requirements of the application scenario. A subscriber targets its subscription when registering a topic of interest to it in a message broker, known as the binding key [47]. The broker then holds a list of binding keys that refer to the subscribers' interests. In addition, each publisher labels each message being published with a topic stored in the message header, named routing key, when sending messages

to a broker. When a broker receives a message, it examines the message header, retrieves the routing key and compares the routing key with the existing binding keys to decide whether some subscribers are interested in this message (matched subscribers). If no matched subscribers are found, messages can be discarded; otherwise, they are forwarded to the matched subscriber(s). Note that in topic-based matching, the broker does not work on the content of each received message. Therefore, topic-based PSMOM has no restrict requirements for the types and the structure of the content data, i.e., both structured (e.g., name-value pair) and unstructured (e.g., pure text string) data of any type, e.g., Text, Byte, Image, and Video, are allowed to be exchanged through a topic-based PSMOM.

In a content-based PSMOM system, the message body of each MOM message needs to follow a pre-defined message schema, which is usually a set of name-value pairs, e.g., *name = XXX, price = XXX, volume = XXX*. Each subscriber targets its subscription with a query to the broker against the message schemas. It is able to set filtering criteria along multiple dimensions [15, 48-53], e.g., *name = "IBM", price < 20, volume > 3000*. Publishers create messages following the schema and send them to brokers. When a broker receives a message, it retrieves the content from the message body, looks up the subscriptions' queries, to decide whether the content of the message satisfies the conditions of any subscription queries. Thus, compared to a topic-based PSMOM, a content-based PSMOM has more restricted requirements on content structure.

A topic-based PSMOM tends to use pre-defined topic names, i.e., once subscribers express their topics of interests, they cannot receive messages for which the message content is relevant, but the topic is different. A content-based PSMOM classifies messages according to the properties of the message content itself. So a content-based publisher/subscriber could be more flexible [28, 54]. However, with the support of filters or pattern matchers, such as the message selector in JMS, a topic-based PSMOM also allows subscribers to express queries to retrieve messages, which match user specified properties defined in the message header. In addition, topic-based matching offers less matching delay and a higher throughput since there is no need to read and extract the content information for each message.

In a TWS, both topic-based and content-based PSMOMs can be adopted for message interaction between distributed system components or processes. However, in TWSs,

there are multiple types of messages that need to be exchanged among different system components, including both structured and unstructured data. For instance, warning messages can be represented using XML extensions such as the Common Alerting Protocol (CAP[3]) format, while the sensor data can be represented using other XML extensions such as the Open Geospatial Consortium Observations & Measurements (OGC O&M[4]) standard, while evacuation advice may be represented as natural language Text. This introduces extra computation complexity to parse heterogeneous content structures when adopting a content-based PSMOM. Thus, the topic-based PSMOM is the focus in this thesis.

2.2.2 Message Broker Architecture

In a PSMOM, brokers from different vendors may use different architectures and use different names for broker components even although they support common broker functions. A broker architecture for Apache Qpid[5] is shown in Figure 2-1 with the following main components: a shared Input Queue for all the messages from publisher, a set of Exchanges that support different types of matching, a Message Store that records the received messages, and a set of Output Queues.

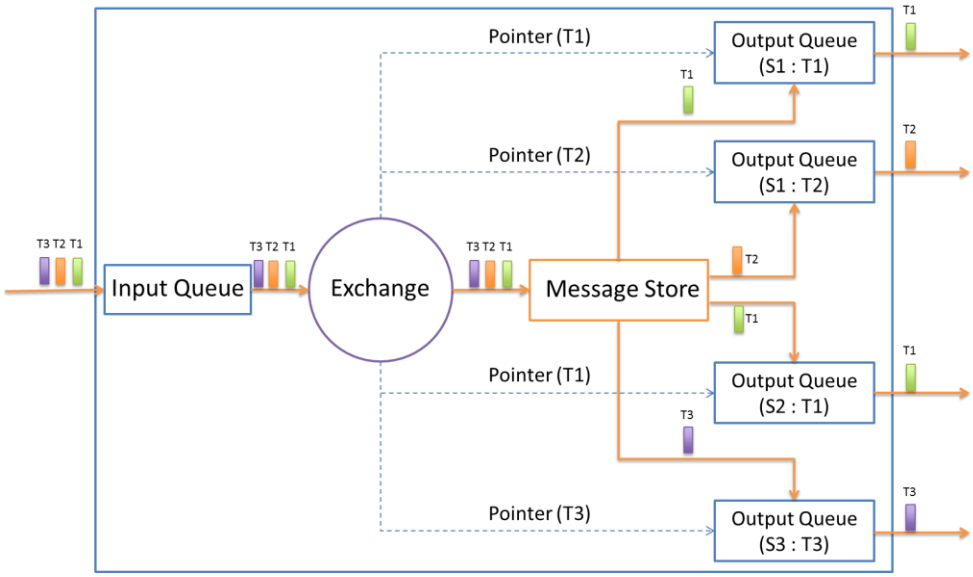

Figure 2-1 Qpid Broker Architecture

[3] CAP - http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html

[4] OGC O&M - http://www.opengeospatial.org/standards/om

[5] Apache Qpid - https://qpid.apache.org/components/java-broker/index.html

In the example shown in Figure 2-1, messages labelled with three different topics (T1, T2 and T3) are published to a broker with three subscribers subscribing to them. Each Output Queue is bound to a subscriber on a specific topic, i.e., subscribers S1 and S2 subscribing to the same topic T1 have different output message queues, e.g., S1:T1 and S2:T1, while a subscriber S1 subscribing to the different topics T1 and T2 has different output message queues, e.g., S1:T1 and S1:T2. The published messages are examined by the exchange component to find a binding to the output queues that correspond to the matched subscribers. The matched messages are recorded in the message store and the pointers to the message store are sent to the matched output queues. Each output queue follows a First-In-First-Out (FIFO) principle. It retrieves the corresponding messages in the message store and forwards them to a subscriber.

2.2.3 Broker Federation

In practice, due to security requirements that require the use of firewalls and restricted IP addresses, geo-distributed publishers and subscribers that exchange messages may only be able to connect to local brokers. For instance, although a publisher in London (P_{London}) and a subscriber in Paris (S_{Paris}) may want to exchange messages with each other, they are not able to as they can only connect to the broker with an IP address in the same country. A single centralized broker design is not feasible in this case. Therefore, broker federation is required, as shown in Figure 2-2.

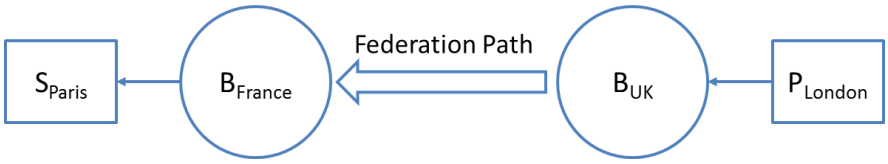

Figure 2-2 Simple Broker Federation

Broker federation allows messaging networks to be defined by creating message routes, in which messages via one broker (the source broker) are automatically routed to another broker (the destination broker). Federated brokers set up federation paths between each other, which allow messages published to the source broker to be automatically routed to the destination broker where the matched subscribers are connected. Normally, a federation path is unidirectional, i.e., it only allows a message to be routed from a source broker to a destination broker. However, a source broker can also work as a destination broker, i.e., if two brokers need to interact with each other,

two federation paths are set up. In the London-Paris communication example above, if a publisher in Paris (P_{Paris}) also needs to communicate with a subscriber in London (S_{London}), an additional federation path from broker in France to Broker in UK needs to be established, see Figure 2-3.

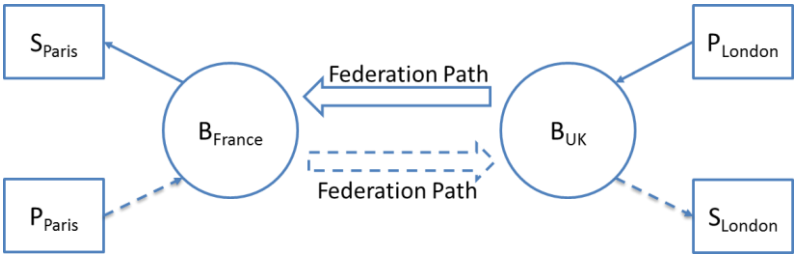

Figure 2-3 Bi-direction Broker Federation

In a TWS, distributed brokers are federated to enable operation centres in different geo-locations to exchange information, e.g., live and historical sensor data and generated workflows, between them for collaborative decision-making.

2.2.4 Comparison of Different Messaging Systems for TWS

There are several candidate messaging system that can provide upstream and downstream communication for a TWS, e.g., RPC, brokerless messaging system, and PSMOM. According to the communication requirements of a TWS specified in Section 1.1, a comparison of these different messaging system is analysed.

RPC is not suitable for a TWS as the data process and crisis prediction takes time, e.g., from a few seconds to a few minutes, which will result in a vital delay in data delivery. A brokerless system is also not a good design choice for a TWS for the following reasons. First, in many-to-many communication, each client in a brokerless system needs to maintain more connections. Second, developing a resilient directory service and queuing system significantly increases the complexity for deploying the brokerless messaging system into a TWS. Third, in a TWS, using a brokerless system design for message exchange is infeasible as some distributed system components cannot communicate directly. Compared to them, a PSMOM is better for deploying in a TWS for the following reasons. First, a PSMOM supports asynchronous information exchange and the clients involved are loosely coupled. Second, a PSMOM reduces the number of connections required in many-to-many communication. Third, a PSMOM

can provide a messaging service for clients that cannot connect directly to each other using a broker federation.

2.3 Broker Overlay

In practice, to enable message exchange between clients in a Wide Area Network (WAN), brokers are interconnected through transport-level links to form a broker federation [55]. In such a multiple-broker publish/subscribe system, brokers are connected in a peer-to-peer fashion to form an overlay network [52, 56].

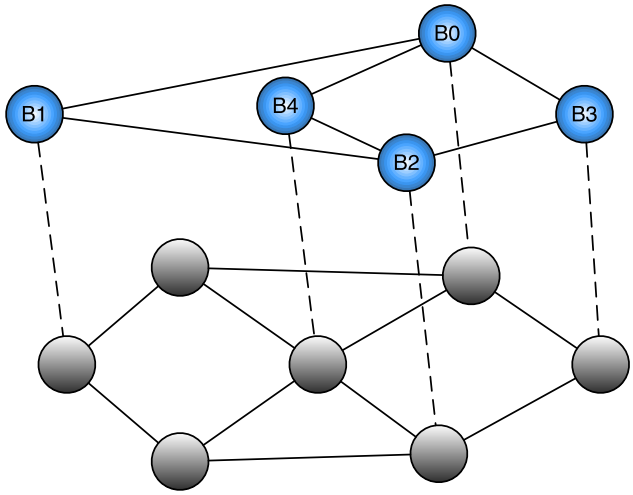

Figure 2-4 Mapping between Broker Overlay and Underlying Network

Figure 2-4 shows an example of a broker overlay consisting of five brokers (B0 to B4) mapped to the underlying network (or physical network). It is shown that the directly connected overlay brokers (e.g., B1 and B2) may not be directly connected into the underlying network. Much research has been done to decide where to deploy the overlay brokers to improve their efficiency and resilience, and to minimize the transmission delay for message dissemination.

A broker overlay network can be statically configured, in which the broker connections, are clearly defined. Therefore, when a publisher or a subscriber joins the system, it is assigned to a broker that exchanges messages according to its topics of interest. The overlay can also be dynamically configured, in which message routes between brokers are not pre-defined but are determined when a new publisher or subscriber joins the system. This can be achieved through *advertising* and *subscribing* processes. *Advertising* is the process to distribute new publishing topic from the source broker, to

which a new publisher connects, to other brokers in the overlay. Thus, all the brokers involved in the overlay have knowledge of the distribution of the published topics, i.e., of the source broker for each topic, and how topics are propagated via the advertising routes. *Subscribing* is the process initialised by a broker, to which a new subscriber connects. It sets up a subscription route to the broker such that any matched publishers can connect to it. In a TWS, the broker assigned to a new publisher or subscriber is affected by several factors, such as the load status of the brokers and the existing client distribution in the broker overlay. Thus, the static configuration method is not applicable in this case. Therefore, a dynamic configuration is required to construct and maintain the broker overlay in a TWS.

2.4 Load Management for PSMOM

Load management has been a widely explored research topic for the past two decades since the introduction of parallel and distributed computing [7]. The goal of load management is to distribute load efficiently to all the available resources in a way that maintains the normal system operation and lowers the risk of overloading individual processing components.

As specified in Section 1.1, brokers in a PSMOM used in a TWS may become overloaded due to a burst of message exchange or due to a reduced system capacity. This would delay the decision-making. Thus, the load management for a PSMOM targets managing the processing and communication load for all the brokers involved. This is achieved with the following processes: load detection, load distribution, load analysis, load migration or offloading, and congestion control. Load detection is the process of detecting the load status of a broker. It is the initial step of load management, which obtains the load conditions for each broker involved in the broker overlay. It always achieved by periodically retrieving a set of pre-defined load metrics, such as CPU and bandwidth usage, which are compared using corresponding thresholds. Load distribution is the process of assigning a broker to a new client (publisher or subscriber) according to the load state of the broker obtained through load detection, the topic information, and the distribution of existing clients. It is an initial attempt to balance the load among brokers in the broker overlay. Load analysis is the process that analyses the load influence for individual message exchange services. It aims to support the

offloading process for client selection and therefore it is sometimes considered as a part of the offloading process. When a broker is detected to be overloaded, offloading is triggered. This process selects and migrates clients from an overloaded broker to one with less load, assuming it has sufficient system resources, i.e., the surplus resource case. However, brokers may not always be available that have surplus resources. A congestion control process needs to be defined to supplement the offloading process to provide load management for the limited resource case. Message congestion occurs when the combined system resources of brokers in a broker overlay are less than those resources required by publishers and subscribers. Two combined pre-conditions trigger the congestion control process. One is that the PSMOM system is in the limited resource case and the other is that the load status of the broker overlay reaches its limit.

To provide load management support for a PSMOM, a Load Manager (LM) is used. This can be either inbuilt into brokers or work as a separate component. LM is often defined as a set of policies for specific load management tasks or processes, e.g., for load detection.

2.5 Summary

PSMOM is widely used to build large-scale distributed systems. It enables messages to be disseminated from publishers to subscribers and decouples message publishers from subscribers in time and space. According to the different matching criteria, PSMOM can be classified into topic-based PSMOM and content-based PSMOM. Although content-based PSMOM offers more flexibility for message matching, it requires the message content to be well structured in order to support content matching. The disadvantages to use content matching for TWS are that there are multiple types of message structure including natural language. The broker federation and the background of the broker overlay are also presented. In practice, a broker overlay is often used in real distributed applications to enable messages to be exchanged between geographically distributed system components that may not be able to communicate with each other directly due to security restrictions such as firewalls. In the next chapter, a critical analysis of existing load management solutions for PSMOMs is presented.

3 LITERATURE SURVEY

3.1 Overview

This thesis focuses on load management for PSMOM used in a TWS. As stated in Section 2.4, the load management process involves load detection, load distribution, load analysis, offloading and congestion control. Therefore, the analysis of the existing methods adopted in each load management process for PSMOM is the focus of this survey. Note that, existing offloading methods take the load analysis process as a part to support client migration, and therefore the analysis of the existing load analysis methods are presented along with the offloading methods (Section 3.4). It assumes that the broker overlay and overlay routing already exist. In addition, as TWS is a distributed system that requires communication in both WAN and LAN environments, other methods that can only apply for a LAN environment are not discussed. The analysed methods are retrieved from either the existing publications or the existing PSMOM systems applied in distributed systems.

3.2 Load Detection

Load detection is the process that detects the state of the broker being monitored, e.g., to determine whether the broker is overloaded, by periodically retrieving some load metrics and comparing the values with corresponding thresholds. In the following subsections, the design for load metrics, load state and load thresholds for existing load management methods for PSMOM are described.

3.2.1 Load Metrics

Different load detection designs may have different load metrics. For a PSMOM, load detection is achieved by retrieving either system level information, or the middleware level information such as matching capacity utilisation.

The system level metric measures some usage of the operation system, such as CPU [57], Memory [58] and Bandwidth [7] that reflects the operational status of the system, on which the broker relies. Some methods have been proposed to map the system level metric such as CPU load, Memory load, and I/O load to the parameters of the message

exchange services, e.g., throughput, number of publishers, and number of subscribers, to measure the capacity of the broker through a set of factorial experiments [59, 60].

Middleware level metrics are used to measure aspects of a broker's function such as matching capacity utilisation, input queue utilisation, output queue utilisation, queue depth, and replication ratio [7, 21]. These values reflect the load status of a broker more specifically, e.g., a 100% used matching capacity means that the broker cannot match any more publishers that join or if their publishing rates increase. Some of these metrics cannot be directly measured in the broker, such as the matching capacity usage, as it requires the knowledge of the maximum matching capacity, which is dynamic. It depends upon the message exchange service and system level metrics. Metrics such as queue depth can reflect an aspect of the broker load status, e.g., a high queue depth value indicates that there are many messages waiting in the queue. However, this does not clearly indicate what caused this load state, e.g., a high queue depth value has many causes, such that the output bandwidth may be already used up or subscribers may be too slow to download messages.

3.2.2 Load State & Load Thresholds

Thresholds are used to help manage broker loads. Each load metric has its corresponding threshold. When the values of load metrics are determined, they are compared to a threshold to classify the load state of the broker into normal or abnormal. Thresholds may be static or dynamic. A static threshold has a fixed predefined value, e.g., 90% CPU usage. Static thresholds do not change during the system operation. They can only be modified if the system is stopped and then restarted. Static thresholds are often used to manage load with respect to system level metrics, such as CPU and bandwidth usage, independent of the dynamic configuration of the middleware, i.e., the number of clients connected and the number of messages exchanged. In contrast, dynamic thresholds can change during the system operation, e.g., a queue depth threshold may be affected by the number of clients being served and by the exchange rate. Dynamic thresholds are often used for the middleware level metrics, which are affected by the operation of the broker. For example, a threshold for queue length, which is used to limit the number of messages held in the queue, may be influenced by the message exchange rate and thus the threshold value needs to be updated during its

operation. For a hybrid load metric design, both static and dynamic thresholds are used for load management, depending on the application specific requirements.

The load state of a broker is classified according to how the values of the load metrics measured relate to their corresponding thresholds. Cheung et. al. define two load thresholds (a high threshold TH_{high} and a low threshold TH_{low}) for each load metric and classify the broker's load state into LOW LOAD, HIGH LOAD and OVERLOAD [7]. The high threshold determines whether a broker becomes overloaded, while the low threshold indicates whether the broker can receive new connections or message exchange requests and whether it can be selected to accept loads from overloaded brokers. The relationship between thresholds and load state for this design is shown in Table 3-1. In addition, a temporary load state BUSY is adopted to indicate that a broker is involved in an offloading process and its load is unstable [7].

Condition	State
$(All\ the\ metrics) < TH_{low}$	LOW LOAD
$TH_{low} < (Any\ Metric)\ \&\ (All\ Metrics) < TH_{high}$	HIGH LOAD
$TH_{high} < (Any\ Metric)$	OVERLOAD

Table 3-1 Relationship between Load State & Load Threshold

3.3 Load Distribution

Load distribution is the process to assign a broker to a new client, e.g., a publisher or a subscriber, according to the load state of the brokers obtained through the load detection process, the topic information, and the distribution of existing clients. This is the initial process that affects the load status of the broker overlay.

Round-Robin (RR) is one of the earliest algorithms proposed for process and network load scheduling in computing based upon equal priority and a circular order [61]. In PSMOM, RR is achieved using three steps. First, a LM generates a list of brokers involved in the broker overlay. Second, when a new client joins the system, a LM picks the first broker in the broker list and assigns it to the client. Third, a LM puts the broker selected at the tail of the list. The load state of each broker and the load usage for each client are not considered, which may cause a highly loaded broker to be overloaded if a new client is still assigned to it. With respect to the differences of load status and

processing capacity for different servers, a Weighted Round-Robin (WRR) method is proposed. With WRR, each server is assigned with a weight. The server with the largest weight has the highest priority to be selected. In PSMOMs that use WRR, when a client joins the system, the broker that has the highest priority is assigned to the client by the LM. Least-N Scheduling (LnS) is one type of WRR for load distribution in PSMOM. Here the term N is a wild card that can be replaced by any property that the system needs to measure to help manage the load distribution. For instance, Least-Connection Scheduling can be applied to PSMOM by assigning new subscribers to the broker that serves the least number of subscribers [62]. Least-Delay Scheduling means that the broker for which the transmission delay between the client and the broker is minimal, has the highest priority to be assigned a new client, e.g., a subscriber [57].

In addition, some researchers introduce subscription based clustering techniques, where the set of subscriptions are partitioned into a pre-defined number of servers or groups (known as clusters) and the subscriptions for the most common topics of interest are assigned to brokers in the same cluster, in order to minimise the total amount of network traffic [63-66]. However, the above work takes no consideration of the load influence of individual subscriptions, which may result in an uneven load distribution if a set of subscriptions has much more traffic than others. What's more, the delay requirements for individual subscriptions have not been considered, which may bring unexpected delays to delay sensitive messaging services.

Correlation-based load distribution is another method that has been used for load distribution [20]. It determines the distribution according to the correlations between different clients with the following principles. First, clients that are highly positively correlated are assigned to different brokers. Second, the highly negatively correlated clients are assigned to the same broker. Third, the average utilisation of broker capacity for all the brokers in the broker overlay is maximized. With this method, the computation of the distribution takes some time, e.g., from minutes to hours, depending on the number of topics and the number of brokers. It works best for the case when all the clients' information, such as the number of publishers and subscribers, message exchange rate have been predefined, and a correlation analysis between clients and topics has been performed, by the LM beforehand. Any update to the broker overlay, e.g., to add or remove a broker, and to client configurations, e.g., add a few new clients,

takes considerable time to re-compute the distribution. Thus, this method cannot be applied to TWS, in which the system configuration and client exchange rate may vary according to the situation of the environment, e.g., the message exchange rate may increase when a tsunami event is detected and a broker's capacity may be reduced due to the damage caused by the aftermath of the event.

Cheung et. al. propose a method that measures the imbalance between brokers to trigger offloading [7, 21]. In such a method, clients are allowed to connect to any brokers in the broker overlay. LM detects the load differences between different brokers, named the imbalance level. If the imbalance level of two brokers exceeds a threshold, offloading is triggered to balance the load between the two brokers, i.e., to migrate some clients in the broker with a higher load to another less loaded one. This type of method increases the flexibility for the load balancing but introduces delays to any client's message exchange that is being selected to move to another broker. Thus, this type of method is not applicable in TWS, in which the delay introduced to time-critical subscription services should be minimal.

In TWS, a hybrid method is required to distribute load in the broker overlay. This needs to be aware of the load status of all the brokers and the delay requirements for different subscription services, and be able to minimise the time delay for time-critical message exchange.

3.4 Offloading

Offloading is the process used to migrate load from an overloaded broker to one with less load. It is triggered when a broker becomes overloaded or when a broker imbalance is detected.

Random offloading is the basic offloading method. As the name suggests, when offloading is required, LM randomly chooses and migrates a client to another randomly chosen broker called the load-accepting broker. This process continues until all the brokers are not overloaded any more. This method takes some time to balance the load in all brokers, as it does not consider the load influence for both the offloading broker and the load-accepting broker. Thus, it may overload the load-accepting broker and requires another offloading process to rebalance the load [7, 21]. Prioritised random

offloading is an improved method in compared with the conventional random one. Instead of randomly choosing a load-accepting broker, LM prioritises such brokers so that a broker with the maximum capacity has the highest priority to be selected as the load-accepting broker. This method therefore reduces the chance that the load-accepting broker becomes overloaded.

Cheung et. al. introduce a load analysis (LA) mechanism to improve the existing offloading methods [7]. The authors state that this is the first load balancing method for PSMOM that adopts load analysis to avoid load-accepting brokers from being overloaded during the offloading process. This LA mechanism prioritises offloading clients and estimates the load influence to both offloading broker and load-accepting broker for each client. Based upon this analysis procedure, LM ensures that the extra load introduced to the load-accepting broker does not exceed its load capacity. The offloading progress is completed as follows. First, the offloading broker locates a load-accepting broker of which the broker's load state is LOW. Second, both the offloading broker and load accepting broker change their state to "BUSY" which indicates they are currently involved in an offloading process, i.e., they will not accept any more work. Third, for each subscription, the offloading broker evaluates the influence to both load-accepting broker and offloading broker based upon the message input rate. Fourth, the system lists all the subscriptions that can reduce the load of the offloading broker and have the least side effects for the load-accepting broker. Fifth, the subscriptions in the list are migrated to the load-accepting broker. Sixth, steps one to five are repeated until the offloading broker is no longer overloaded. However, this method does not consider the different transmission delay requirements for different subscriptions services and this therefore may introduce unnecessary delays to delay sensitive services.

3.5 Congestion Control

Congestion control is a widely explored topic in Internet Protocols [8, 67], Wireless Sensor Networks [68, 69], Interconnection Networks [70], Opportunistic Networks [71], and Multimedia Streaming Networks [72]. In general, congestion occurs when the traffic load exceeds the network capacity. This may result in packet loss and increase queuing delays, and lead to retransmission that consumes extra energy. This especially affects bandwidth intensive and delay sensitive applications, and applications affected

by message loss. Much work has been proposed to resolve the problems, such as priority based congestion control [73], topology-aware resource adaptation [74], predictive congestion control [75], and receiver assisted congestion control [76].

Different congestion control mechanisms have two similar phases: congestion detection and congestion handling. In the congestion detection phase, the load states of brokers are periodically detected to determine whether congestion occurs. In the congestion-handling phase, congestion-handling policies are applied to maintain the overall system performance and QoE for subscribers. For a PSMOM, congestion occurs when the processing and transmission capacity of brokers or links to brokers are used up. Existing solutions that handle congestion in the broker overlay can be categorised into two approaches: publishing rate control [8, 56] and path handling [77, 78].

3.5.1 Congestion Control by Publishing Rate Control

A congestion control mechanism, which handles congestion for a PSMOM by controlling the publishing rate, is described in [8]. The authors specify three types of brokers: publisher hosting brokers (PHB), intermediate brokers (IB), and subscriber hosting brokers (SHB). Each PHB hosts one or more publishing endpoints, referred to as pubends (PE). Each pubend represents an ordered stream of messages published by one or more publishers, and maintains this stream in persistent storage. Messages published from different publishers may be assigned to the same pubend. This pubend decides on a position for the message in the persistent stream and logs the message to a persistent store. After that, the pubend sends the message towards SHBs through IBs. The IBs forward data and control messages to the SHBs. Figure 3-1 shows the connection between PHB, IB and SHB.

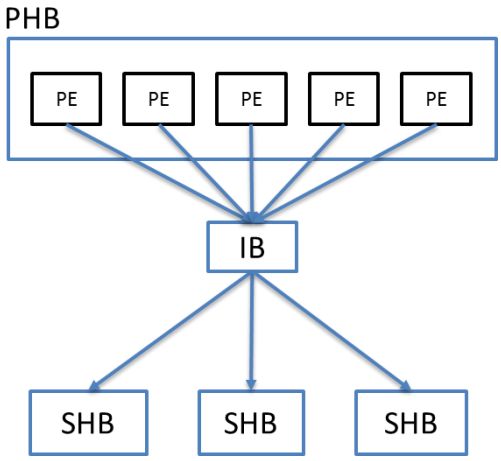

Figure 3-1 Interaction between PHB, IB and SHB

Based upon the above model, the authors propose two congestion control protocols, a PHB-driven congestion control (PDCC) protocol and a SHB-driven congestion control (SDCC) protocol. PDCC regulates the rate at which new messages are published by a pubend. The publication rate is adjusted depending on the observed throughput at the SHBs. It is the responsibility of the SHBs to calculate their own congestion metric based on throughput and notify the pubends whenever there is a risk of congestion. In this design, SHB use the ratio of pubend to SHB message rate as a metric for detecting congestion, i.e., $r_{pubend} / r_{SHB.}$ SDCC manages the rate at which an SHB requests missed data by sending NACKs upstream. These protocols are implemented into gryphon[6] brokers and the experiment results show that the proposed PDCC and SDCC protocols can maintain the system performance by preventing the system from becoming congested.

The proposed congestion control mechanism has some limitations. First, it is not designed for fully loaded brokers but is designed to provide an acceptable level of performance for slow subscribers and for the situation when the system is recovering from link failures. Both PDCC and SDCC actually increase the load of PHB and IB, as they need to hold more messages in the message queue before sending them along the next hop. This introduces unexpected delays to delay sensitive messages. Second, the proposed methods ignore the importance of the messages to the matched subscribers in terms of controlling publishing rate. Therefore, these methods may reduce the QoE for

[6] Gryphon - http://www.research.ibm.com/distributedmessaging/papers/ext-abstract.htm

subscribers that exchange the more important messages. Third, the proposed solution makes modifications to the broker internals. This increases the maintenance and update complexity when new versions of brokers are released by vendors as their congestion control mechanisms need to be re-integrated and rebuilt.

3.5.2 Congestion Control by Path Handling

A congestion control mechanism that handles congestion using path handling is described in [77]. For this proposed algorithm, the base assumption is that in a congested PS system, it is not possible to provide unaffected services for everyone all the time. Therefore, in this approach, a priority-based method is integrated seamlessly with the PS system, without violating the PS decoupling. In this design, the authors consider two congestion situations, i.e., a broker is congested, or the links to the broker are congested. To handle both congestion situations, brokers are modified. Thus, they can skip links when a broker itself is congested or drop less profitable messages for the system when links to a broker become congested. To detect the profitability of subscriptions, each subscription message in this system uses a maximum price and coverage metric. Maximum price shows the value that this message has for a subscriber. Coverage is used to determine how many subscribers receive a publication message on a given link. The profit of a message is proportional to the value of the maximum price and coverage metric. Whenever a broker is congested, it skips any message queues that contain the least profitable subscriptions. Whenever the situation returns to normal, only then will it process these queues. If the broker becomes too congested, queues build up and the broker runs out of memory and eventually crashes. For the case when one or more links to a broker are congested, the broker drops the messages with the less profitable information; it only disseminates messages through congested link(s) that have a higher profit.

Another congestion control mechanism for path handling in a PSMOM can be found in [78]. This focuses on a scenario where only some specific brokers and links are congested in the broker overlay. The congestion handling is achieved by distributing the higher traffic of congested brokers and links to other parts of the broker overlay that are not congested. In this design, each congested broker or link has a list of alternative brokers and links. When a broker or link becomes congested, the immediate senders,

e.g., publishers or brokers that connect to the congested broker, connect to an alternative broker through an alternative link.

The proposed path-handling congestion control methods have some common limitations. First, these methods target handling congestion when only some of the brokers and links are congested, i.e., they are not designed for the case when all the available communication and computation resources are used up. Second, for path handling, the load influence on the alternative broker and link is not considered, hence, the selected alternative broker and link may become congested. Third, the proposed solutions make modifications to brokers to enable path handling, and thus increase the complexity in maintaining and updating the system.

3.6 Summary

In this chapter, a comparative analysis of existing methods for load detection, load distribution, offloading and congestion control is given. The load analysis process is considered as part of the offloading process to support client selection for migration. According to the communication requirements specified for TWS in Section 1.1, existing methods need to be improved in the following ways in order to be applied for TWS. First, to detect load status of broker in TWS, hybrid load metrics including both system level metrics such as Bandwidth and middleware level metric such as capacity utilisation need to be adopted. Second, the load distribution process should not only be aware of the load status of the broker, but should also aim to reduce the network traffic and to minimize the transmission delays. Third, the load analysis and offloading process should take the delay requirements for different subscription services into consideration in the client selection process in order to avoid introducing unexpected delays to delay sensitive services. Fourth, the congestion control methods used need to be aware of the importance of messages when discarding messages to reduce load to brokers for the limited resource case.

In the following chapter, Chapter 4, a load management framework that extends the PEER framework, named ePEER, is proposed to manage broker load for PSMOMs in TWS. The construction of the broker overlay for TWS, and the processes for load management for surplus resource case, including load detection, load distribution, load analysis, and offloading, are described. Then, chapter 5 provides a feedback driven

congestion control model as a supplementary load management method for the limited resource case.

4 DELAY REQUIREMENTS DRIVEN LOAD BALANCING FOR SURPLUS RESOURCE CASE

4.1 Overview

In this chapter, an overview of the load management framework, ePEER, is presented. This includes a detailed description of a head-edge (H-E) broker overlay design that supports the message dissemination requirements for TWS and the design of the LM, or Management Agent, used to manage the workload of the H-E broker overlay in both the surplus and limited resource case. In addition, the delay requirement driven load balancing (DRD-LB) method for ePEER is proposed. DRD-LB manages the load among the H-E broker overlay for the surplus resource case in order to reduce unexpected delays introduced to time-critical subscription services. A comparison between DRD-LB with the state of the art load balancing method adopted by PEER [7], denoted as PEER-LB, is also described to emphasise the benefits of ePEER.

4.2 Head-Edge Broker Overlay

The H-E broker overlay is partitioned into several local domains, named clusters. In each cluster, there is one head broker and several edge brokers to form a tree hierarchy. In each cluster, the cluster-head (head) broker can have more than one neighbour while a cluster-edge (edge) broker has only one neighbour broker. In addition, in each cluster, all the publishers connect to the head broker while all the subscribers connect to the edge broker. Thus, all the messages are routed from the head broker to the edge broker. Figure 4-1 shows an example of an H-E cluster with one head broker (H) and three edge brokers (E1, E2 and E3). In this example, three matched publisher-subscriber pairs, i.e., *P* and *S*, *P'* and *S'*, and *P''* and *S''*, exchange messages within the cluster, i.e., use intra-cluster communication.

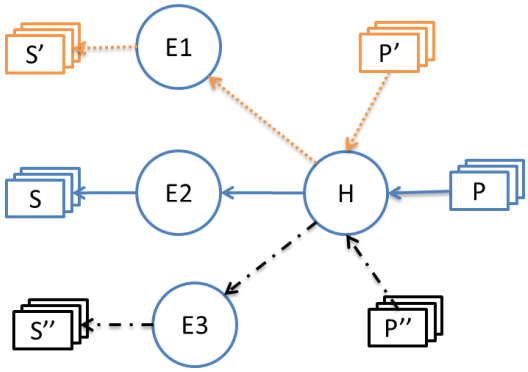

Figure 4-1 An H-E Cluster with One Head (H) and Three Edges (E1 – E3)

In addition, cluster-to-cluster (inter-cluster) communication occurs via federation paths created between different head brokers. Figure 4-2 shows an example of inter-cluster communication between two H-E clusters. In this example, the publisher *P''* in cluster 1 has a matched subscriber *S''* remotely hosted in cluster 2. After the advertising and subscribing processes (Section 2.3), the messages published in cluster 1 are then routed to head broker H' of cluster 2 and then forwarded to edge broker E2', to which *S''* is connected. This also applies to matched publisher *P'* and subscriber *S'*.

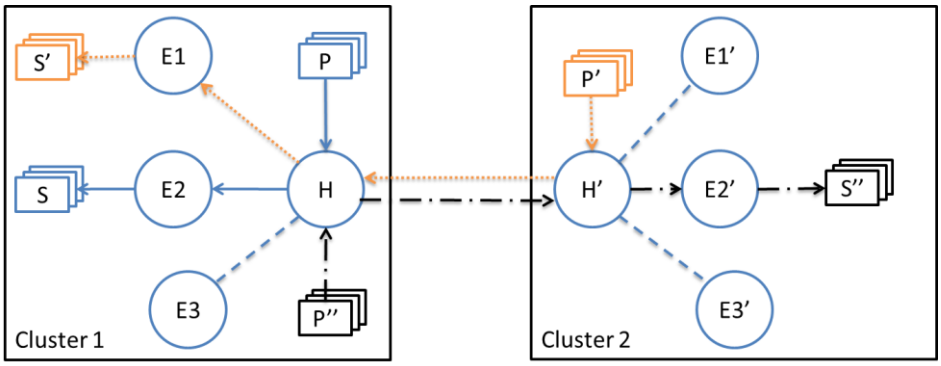

Figure 4-2 Inter-Cluster Communication

The benefits of adopting such an H-E broker overlay design to support message dissemination for TWS are as follows. First, H-E broker overlay design allows all the brokers in a cluster to be deployed in a LAN environment, e.g., in a warning centre. It then enables messages to be disseminated within one warning/data centre (intra-cluster) for processing and among multiple warning/data centres (inter-cluster) for collaborative decision-making. Second, the H-E broker overlay also allows brokers in a cluster to be deployed in a WAN environment, e.g., in the offshore area that is potentially affected by Tsunami. This can also be used to disseminate information from warning centres to registered stakeholders in such an area. Third, since a head broker only serves the publishers, and an edge broker only serves the subscribers, the client selection process

for offloading is simplified, i.e., when a head broker becomes overloaded, only publishers are required to be analysed and migrated; when an edge broker becomes overloaded, only subscribers are offloaded. Fourth, the federation setup process is simplified, as all the messages being disseminated within a cluster are always routed from head broker to edge broker, which means that the federation path within one cluster is unidirectional. Fifth, regarding the scalability and resilience requirements of a TWS, the H-E broker overlay can be extended to add fault tolerance, e.g., mirroring can be used to ensure published messages to be received by matched subscribers in the case of a broker or link failure [9, 18, 19, 79].

As specified in Section 1.1, to detect a tsunami for a particular region, several national warning centres and one regional warning centre work collaboratively to detect tsunamis and disseminate warning messages to the affected region. In each warning centre, an H-E broker overlay is deployed. A federation path between two head brokers is established only when communication occurs between two centres. Figure 4-3 shows an example of how clusters are organised in a TWS. NTWC and RTWC refer to the cluster deployed in a national tsunami warning centre and regional tsunami warning centre respectively. Loc. refers to the cluster deployed in a region monitored by a warning centre. The region could be one country or several countries in practice.

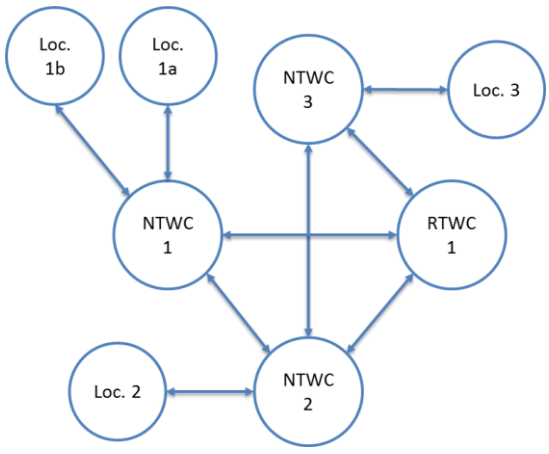

Figure 4-3 Clusters Organization in a TWS

As shown in the diagram, the broker overlay for this TWS is designed as follows:

1. Each cluster follows the H-E broker overlay design, i.e., there are one head and several edges. The head and edge brokers are federated unidirectionally, i.e., messages are only routed from head to edges.

2. Different clusters are federated through head brokers.
3. Each cluster has a set of pre-defined neighbour clusters that are directly connected:
    a) Between national tsunami warning centre and the regional tsunami warning centre, e.g., between NTWC1 and RTWC1;
    b) Between national tsunami warning centre and its observation region, e.g., between NTWC1 and Loc.1a;
    c) Between national warning centres that need to share information for decision-making, e.g., between NTWC1 and NTWC2.
4. When messages needs to be disseminated between clusters that are not directly connected, another message dissemination route needs to be determined according to the advertising and subscribing process described in Section 2.3.

4.3 Distributed Management Agent

As specified in Section 2.4, a LM is required to provide load management for PSMOM. LMs can be classified into internal and external to the broker. An internal LM requires modifying brokers to support load detection, analysis, and offloading functions; an external LM is an extra component, which is not part of a broker but able to monitor and analyse the load state of broker via a message-based network link. The former method requires a modification to the brokers and therefore is harder to maintain than the latter one. In the thesis, an external LM to the broker named a management agent (MA) is used. For the H-E broker overlay, each broker is managed via an MA. The MA that manages a head broker is named a Head MA (HMA), while the MA that manages an edge broker is named an Edge MA (EMA). It is the same as for the H-E broker overlay: an EMA is only allowed to communicate with HMA of its cluster, while an HMA is able to communicate with all EMAs in its cluster and with the HMAs of other clusters. Communication between MAs is accomplished using MOM messages, exchanged through the management brokers. The management brokers that manage the message exchange run aside from their corresponding head or edge brokers. For example, the management broker used by the EMA of an edge broker E1, e.g., $E1_m$, runs on the node that hosts E1. In this thesis, the basic broker used is an open source release of Qpid (version 0.18) that speaks AMQP. Therefore, in this design, all the message exchange processes through a broker follow a subset of AMQP, as some of

AMQP functions such as type-based matching and subject-based matching are considered irrelevant. The broker architecture is described in Section 2.2.2.

4.3.1 Components of Management Agent

Table 4-1 shows a summary of how HMA and EMA perform in the load management life cycle, i.e., load detection, load distribution, load analysis, and offloading.

<table>
<tr><th></th><th>HMA</th><th>EMA</th></tr>
<tr><td rowspan="4">Load Detection</td><td>Periodically detects the load status of head Broker</td><td rowspan="4">Periodically detects load status of the corresponding edge broker</td></tr>
<tr><td>Collects and records the load status of all edge brokers in the same cluster</td></tr>
<tr><td>Calculates the cluster-level load status and shares it among neighbour clusters</td></tr>
<tr><td>Records the load status of the load in neighbouring clusters</td></tr>
<tr><td>Load Distribution</td><td>When it receives a client registration request, it assigns a head broker to a publisher and an edge broker to a subscriber following a distribution policy (Section 4.4.1)</td><td>N/A</td></tr>
<tr><td>Load Analysis</td><td>Estimates the load influence of publishers</td><td>Estimates the load effects of subscribers</td></tr>
<tr><td rowspan="3">Offloading</td><td>When head broker becomes overloaded, it locates the load-accepting broker and then selects and migrates publishers to it</td><td rowspan="3">When an edge broker becomes overloaded, it locates the load-accepting broker, selects clients for migration and sends a request to its HMA</td></tr>
<tr><td>When offloading request is received from an EMA, it updates the load state of both overloaded and load-accepting edge brokers</td></tr>
<tr><td>Notifies a selected subscriber to migrate to the load-accepting edge broker</td></tr>
</table>

Table 4-1 HMA and EMA for Load Management in H-E Broker Overlay

As is specified in Table 4-1, HMA performs additional functions compared to EMA. However, in practice, each edge broker and EMA are designed to be able to become a

head broker and HMA respectively when the original head becomes unavailable[7]. Therefore, HMA and EMA have the same load management components but these components perform different functions with respect to the types of the MA. Figure 4-4 shows an example of the load management components for HMA and their interaction. The functions of each component for both HMA and EMA are described in details as follows.

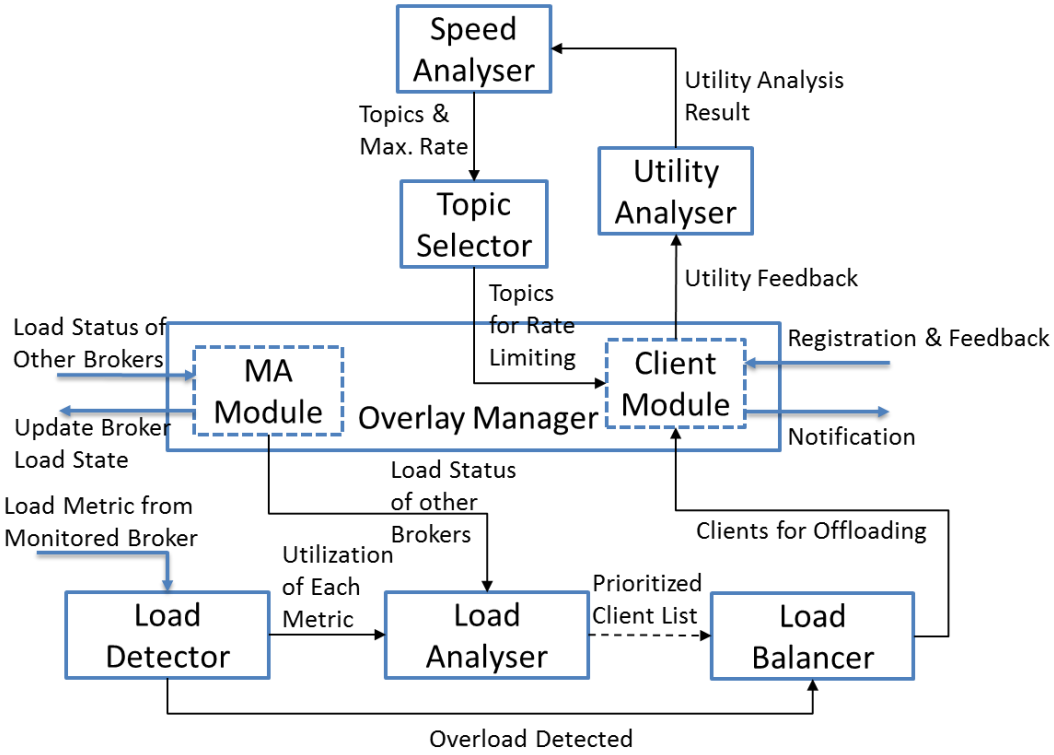

Figure 4-4 Load Management Components for an HMA

The Overlay Manager (OM) has two modules, named MA module and client module. The MA module is used for both EMA and HMA, which is used to exchange information between MAs. It provides the following functions. First, it is used for EMAs of the same cluster to update the load status of corresponding edge brokers to the HMA, so that the HMA obtains the load status of all the edge brokers in the cluster. Second, it is used by a HMA to forward the load status update from an EMA to the other EMAs in the same cluster, so that all the EMAs in the same cluster have the load status of all the edge brokers. Third, it is used by a HMA to receive load status update

[7] The switchover function is considered out of the scope for load management and thus the detail is not described.

from HMA of neighbor clusters and to notify the HMAs of other neighbor clusters. The client module is only used by HMA, which is used to communicate with publishers and subscribers. Through the client module, HMA receives registration request from both publishers and subscribers, assigns brokers to them, receives feedbacks from subscribers, notifies them to migrate from one broker to another, and informs the publishers to publish less messages.

Load Detector (LD), Load Analyser (LA), and Load Balancer (LB) work together to detect and balance the load between brokers, i.e., to provide DRD-LB for H-E broker overlay. For both HMA and EMA, LD periodically retrieves the broker's load information to detect the load state and reports this to its LA (Section 4.4.1). In addition, for LD in EMA, it notifies OM to update the monitored load state to an HMA. HMA then obtains the load status of all the edge brokers, which is the basis of the load distribution process (Section 4.4.2). A LA of a HMA profiles the load distribution for publishers. A LA of an EMA profiles the load distribution for subscribers (Section 4.4.3). Whenever an overload is detected, a LD invokes the LB to start to balance the load, via offloading (Section 4.4.4). The main difference between LB in a HMA and an EMA is that the latter one does not notify the OM to migrate selected subscribers but to update the offloading clients' information to the HMA to let HMA starts the migration. The main reason for this design is to make the client-MA interaction simpler, i.e., both publishers and subscribers only interact with HMA. The details of the load detection, load distribution, load analysis and offloading processes are described in Section 4.4.

Utility Analyser (UA), Speed Analyser (SA), and Topic Selector (TS) are only used by HMA to supplement the DRD-LB method by providing a FDCC support to manage load in the limited resource case. UA analyses the utility of publishers, utility of topics, and importance of topics based upon the utility of messages measured by the matched subscribers. SA computes the max publishing rate for each publisher according to its utility value. TS is invoked when there is no available capacity to balance load for the overloaded broker in the entire PSMOM system. This congestion situation is detected via the LD component. TS then limits the publishing rate to any publisher for any publishing topic according to its importance value. The rate limit information is sent

through the OM to the corresponding publishers. The details of this FDCC model are described in Chapter 5.

4.3.2 Construction of a Head-Edge Broker Overlay

Another important function of an MA is to construct the H-E broker overlay. An OM is able to receive client registrations and communicates these with an OM in another MA. The H-E broker overlay is then constructed based upon this.

Each MA is designed to manage the life cycle of its corresponding brokers, i.e., the management broker used to exchange control messages between MAs and the data broker used to exchange data messages. The reason to introduce an extra management broker is to reduce the load caused by exchanging control messages, such as client migration notification and offloading request, to the data broker. Each MA starts and can restart the brokers using command scripts. In addition, as MAs have the same interaction restriction as the H-E broker overlay, an EMA needs to know the information of the management broker of the HMA, while a HMA needs to know the information of the management broker of HMA in neighbouring clusters. Such information is pre-defined in an MA configuration file and is retrieved by MA when it is initialised.

The role of broker and the interlinked state are specified in a configuration file. The "*brokerType*" property defines the type of a broker, e.g., a head broker or an edge broker, as well as the type of the MA, i.e., HMA for head broker and EMA for edge broker. For example, in a configuration file, if the value of "*brokerType*" property is *head*, it means that when any MA initiated using this configuration file, it behaves as an HMA. The HMA then starts the head broker and head management broker using the broker information list in the configuration file. Similarly, if the value of the "*brokerType*" property is *edge*, it means that the related MA and broker are EMA and edge broker respectively. In an EMA configuration file, it is also necessary to specify the IP address and port of its corresponding management broker for HMA. In an HMA configuration file, the HMA management broker information regarding neighbouring clusters is also included. In order to understand the configuration file better, and how to update it, an example file for an HMA is given in Figure 4-5.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>This file contains the data that allows elements in the system to
know where to find other elements in the system.</comment>
    <entry key="clusterID">WP7_dataCenter</entry>
    <entry key="mode">H-E</entry>
    <entry key="brokerType">head</entry>
    <entry key="brokerIP">138.37.94.94</entry>
    <entry key="brokerPort">5672</entry>
    <entry key="brokerJMX">8999</entry>
    <entry key="brokerMgrPort">5673</entry>
    <entry key="numOfNeighbours">1</entry>
    <entry key="neighbourDomain1ID">WP7_drillingSite</entry>
    <entry key="neighbourDomain1HeadBrokerIP">138.37.94.93</entry>
    <entry key="neighbourDomain1HeadBrokerMgrPort">5673</entry>
</properties>
```

Figure 4-5 Configuration File for an HMA

As shown in Figure 4-5, there are two brokers running in the node with IP 138.37.94.94: a head broker and a head management broker. The management broker is used to exchange control messages between the HMA and EMAs in the same cluster, between HMA and publishers and subscribers in the same cluster, and between HMA and HMAs in neighbour clusters. When an HMA, denoted as HMA_s, is initialised with this configuration file, it first loads the broker information from its configuration file and starts the corresponding head broker and head management broker. It then retrieves the information of management head broker for any neighbouring cluster and publishes a registration MOM message, which is used to notify a neighbouring HMA, denoted as HMA_n that it has gone online. This registration process follows a request/reply interaction pattern, which means that HMA_s waits for a confirmation messages from HMA_n within a given time period. If the confirmation message is received within this period, HMA_s records the state of the neighbour cluster as online; otherwise, the state is recorded as offline. Alternatively, when a HMA_n receives a registration message, it records the state of the cluster where HMA_s belongs to, as online, and publishes a registration confirmation message back to the HMA_s to indicate that the registration message is received and the state is updated. With this step, both HMA_s and HMA_n obtain the information that the neighbour cluster has gone online. It is then able to exchange messages such as publishing advertisement messages.

Figure 4-6 gives an example of an EMA configuration file in the same cluster. The configuration file shows that the management broker of HMA locates in the same cluster as the EMA, i.e., it runs in a server with an IP address of 138.37.94.94. The port

used by the management broker of HMA is 5673. Meanwhile, an edge broker and an edge management broker are running with an IP address of 138.37.94.90. The edge management broker is used for message exchange between EMA and HMA. After the EMA is started, it first sends a registration message to the HMA of its cluster following a similar request/reply approach as described above for HMAs.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>This file contains the data that allows elements in the system to
know where to find other elements in the system.</comment>
    <entry key="clusterID">WP7_dataCenter</entry>
    <entry key="mode">H-E</entry>
    <entry key="brokerType">edge</entry>
    <entry key="headBrokerIP">138.37.94.94</entry>
    <entry key="headBrokerMgrPort">5673</entry>
    <entry key="brokerIP">138.37.94.90</entry>
    <entry key="brokerPort">5675</entry>
    <entry key="brokerJMX">9002</entry>
    <entry key="brokerMgrPort">5676</entry>
</properties>
```

Figure 4-6 Configuration File for an EMA

After all the MAs have started, each MA has the knowledge of their neighbouring MAs, based upon these configuration files. It is assumed that the broker overlay remains stable using standard fault-tolerance techniques such as broker mirroring and link replication.

When a new client joins the system, i.e., it registers with the OM of HMA. OM then follows the advertising and subscribing processes described in Section 2.3 to set up any federations as required. For example (Figure 4-3), when a publisher in RTWC1 needs to publish warning messages and there are subscribers in Loc. 1a that subscribe to the same topic, the message dissemination route is established as follows.

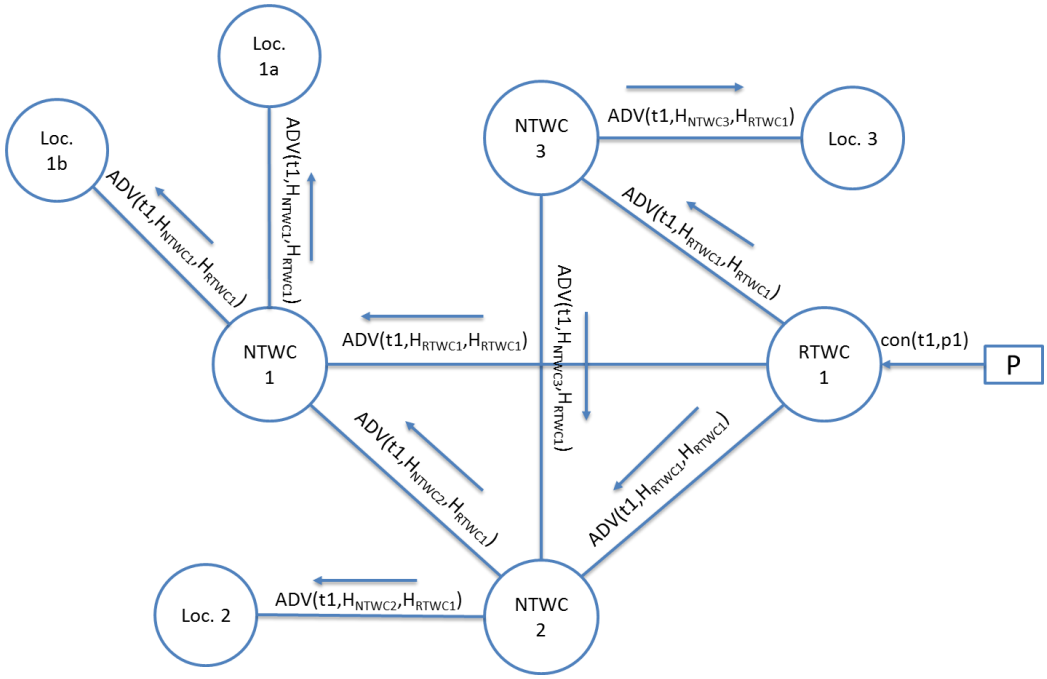

Figure 4-7 Advertising Process

Figure 4-7 shows an example of the advertisement process. After all the clusters have been initialised, when publisher *p* with unique ID *p1* and topic *t1* registers with the HMA of RTWC1, the HMA checks its local publishing topic list (PTL) to detect whether the topic has been advertised. If the topic is a new one, the HMA then records the topic into the PTL and starts to advertise the topic information to all its neighbours (HMAs of NTWC1, NTWC2, and NTWC3) with an advertisement message (ADV). The ADV includes the topic information, the ID of the head broker HMA that starts the advertisement (e.g., H_{NTWC3}, called the bypass broker), and the source broker, to which the publisher connect (e.g., H_{RTWC1}). When an HMA, e.g., HMA of NTWC3, receives the advertisement, it checks the topic and source broker information in its local advertisement table (ADT), to determine whether this ADV has been received. If such information for the received ADV is the same as that recorded, this ADV is marked as an "old" one and discarded. If this ADV is a new one, the HMA then records the advised information into the ADT. It further generates a new ADV that replaces the bypass broker information with the local head broker, e.g., it updates the *ADV(t1, H_{RTWC1}, H_{RTWC1})* to *ADV(t1, H_{NTWC3}, H_{RTWC1})*, and sends this new ADV to the HMAs of their neighbours, except to the HMA, from which the original ADV is received, e.g., HMA of RTWC1. Each HMA involved in the advertising process follows the above step to spread the ADV to its neighbours. For any HMA that receives an ADV more than once,

(e.g., HMA of NTWC2 in the above example), they ignore the duplicate one because it has a higher transmission delay. Figure 4-8 presents pseudo code of how an HMA performs when an advertisement message is received.

```
-------- HMA that receives registration from a new publisher -------------
BEGIN
Let "t" = the publication topic of the new registered publisher
Let "h_p" = the head broker assigned to the publisher
If "t" is a new topic
Then
     Set ADV message with topic = "t", broker_bypass = "h_p", broker_src = "h_p"
     For each HMA of neighbour cluster (nHMA)
          HMA publishes the ADV message to nHMA
     End Loop
Endif
END

----------- HMA that receives ADV messages from neighbour HMA ------------
BEGIN
Let "ADV_recv" = ADV message received
Let "t" = topic recorded in "ADV_recv", "b_bypass" = broker_bypass recorded in "ADV_recv", "b_src"
= broker_src recorded in "ADV_recv"
Let "h" = the head broker of this cluster
If the advertisement table "ADT" contains "t"
Then
     Retrieves existing source broker from the records (b_src')
     If "b_src" is different from B_src'
     Then
          ADT updates the existing record with "t" by adding "b_bypass" and "b_src"
     Else
          Goto END
     Endif
Else
     ADT adds a new record with "t", "b_bypass, and "b_src"
Endif
Set "ADV_out" = "ADV_recv(t,h,b_src)"
For each nHMA
     Let "head" = the head broker of the nHMA
     If "head" is different from any B_bypss in the ADT records for topic "t"
     Then
          Publishes "ADV_out" to nHMA
     Else
          Continue
     Endif
End Loop
End
```

Figure 4-8 Pseudo Code of Advertising Process

After this advertising process, the ADT of each HMA has been updated. It means that the neighbour clusters involved in the broker overlay is able to route messages under the advertisement topic *t1*. In addition, the PTL of HMA in RTWC1 has recorded the

topic *t1* that has been advertised, while the PTLs of the remaining brokers are empty. The PTL and ADT information of the broker overlay is shown in Figure 4-9.

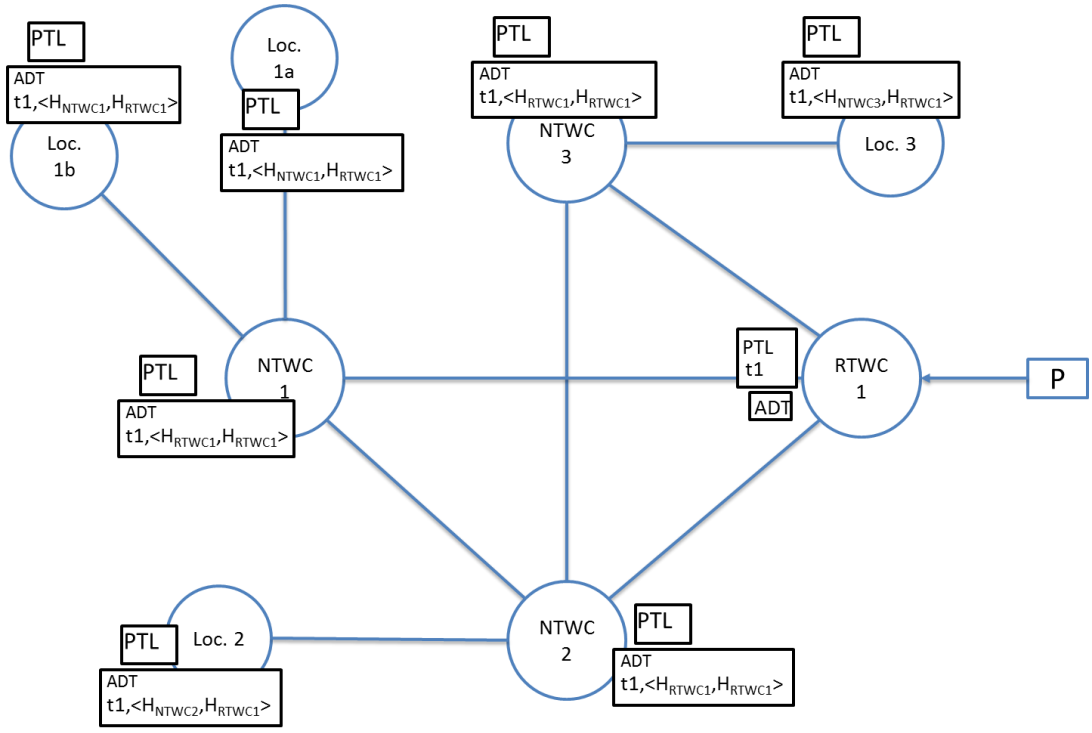

Figure 4-9 PTL & ADT of Broker Overlay after Advertisement

When a subscriber *s* that subscribes to topic *t1* joins the system, e.g., it registers with HMA belonging to Loc. 1a, the HMA checks its ADT and notifies the head broker $H_{loc.1a}$ to set up a subscription to its neighbour head broker that advertises topic *t1* to this cluster, e.g., H_{NTWC1}. This subscription is set along the reverse advertisement path and reaches H_{RTWC1}, to which the matched publisher is connected.

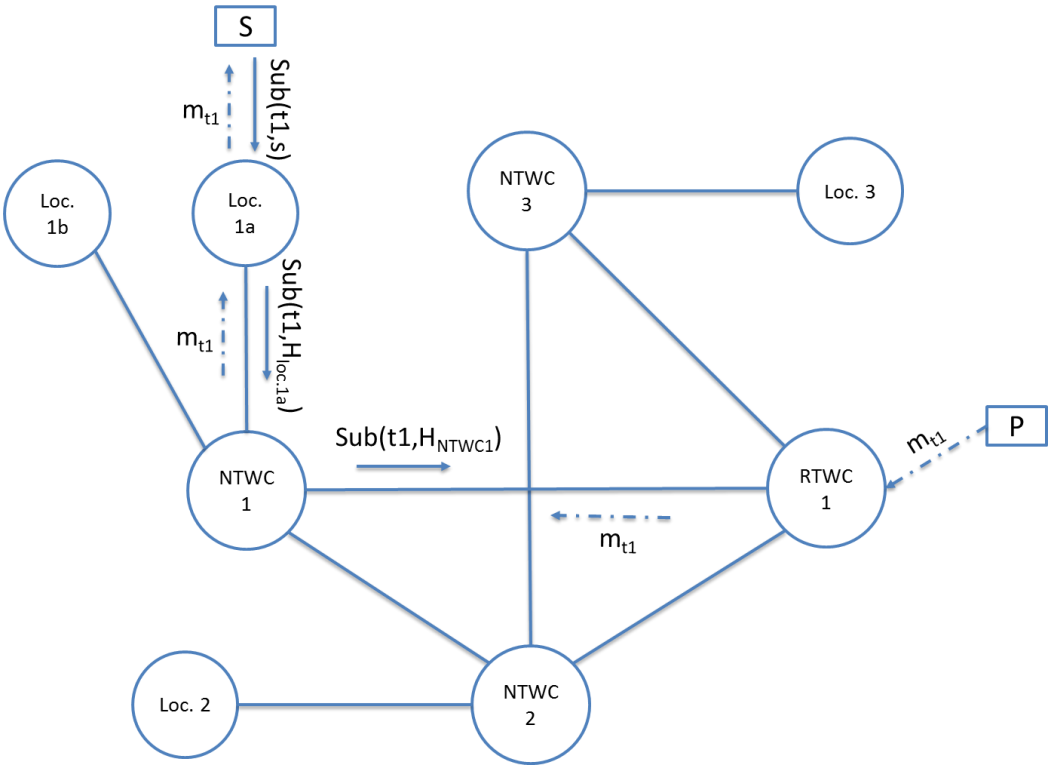

Figure 4-10 Subscribing Process

Figure 4-10 shows an example of subscribing process. In this figure, the solid arrows indicate the subscription flows, i.e., from *s* to the edge broker in cluster Loc. 1a, denoted as $E_{loc.1a}$, from $E_{Loc.1a}$ to $H_{Loc.1a}$, from $H_{Loc.1a}$ to H_{NTWC1}, and from H_{NTWC1} to H_{RTWC1}. The subscribing process takes place for two difference cases. First, an advertisement message is received, in which the topic has been subscribed by some local subscribers. Second, a new subscriber joins the system and subscribes to a topic that has already been advertised by other HMAs. The procedure for both cases are the same. The following pseudo code describes the new subscriber case as an example to illustrate how the subscribing process works, as shown in Figure 4-11.

```
----- HMA receives a registration from a new subscriber -----
BEGIN
Let M_r = registration message received by an HMA
Let t = the topic of the registered subscriber retrieved from M_r
Let E = the edge broker assigned to the subscriber
Let H = the head broker of the cluster
Let T<topic,Edge> = a map that records the topic to Edge information
Update T with a new record <t,E>
Set a new federation F<H,E,t>
If ADT contains "t"
Then
     Let S_bypss = the set of B_bypass for topic "t"
     For each B_bypass in S_bypass
          Let nHMA_b = the corresponding HMA for B_bypsss
          Let M_req<B_req,B_src,topic> = the subscription request including the request
          broker, source broker and topic
          Sends M_req<H,B_bypass,t> to nHMA_b
     End Loop
Endif
End

---- HMA that receives subscription request from neighbour HMA----
BEGIN
Let M_req = the subscription request message
Let t_req = topic retrieved from M_req
Let B_req = request broker retrieved from M_req
Let B_src = source broker retrieved from M_req
Let H = the head broker of the cluster
If B_src == H
Then
     Set a new federation F<H,B_req,t_req)
Else
     Set a new federation F<H,B_req,t_req)
     If ADT contains "t_req"
     Then
          Let S_bypss = the set of B_bypass for topic "t_req"
          For each B_bypass in S_bypass
               Let nHMA_b = the corresponding HMA for B_bypsss
Let M_req'<B_req,B_src,topic> = the subscription request including the request broker,
source broker and topic
Sends M_req'<H,B_bypass,t> to nHMA_b
          End Loop
     Endif
Endif
End
```

Figure 4-11 Pseudo Code of Subscribing Process

Other message dissemination routes are also established by following the above steps. The messages exchanged between MAs for advertising and subscribing processes use the standard broker publish/subscribe paradigm, i.e., an MA acts as a publisher when it advertises new topics to any neighbour MAs and acts as a subscriber when it receives advertisements from any neighbour MAs. Similarly, when a new subscription needs to be propagated through the overlay, the source MA acts as a publisher to update the

subscription information to the neighbour MA that has advertised the corresponding topic. By following the above steps, the broker overlay and all its federation paths are set up.

This H-E broker overlay construction method also provides the opportunity for adding a new edge broker or cluster after the overlay has been constructed. EMA of the new edge broker or HMA of the new cluster only needs to send registration information to the corresponding HMAs. The limitation of this method is that it requires the broker connection information stored in the configuration file to be valid. Any update to the configuration file requires a restart for the corresponding MA and its brokers. Therefore, to ensure the broker overlay construction are valid, some overlay model checking algorithms or tools [80-82] can be applied.

4.4 Design of DRD-LB

A DRD-LB tends to balance the load for the H-E broker overlay in the surplus resource case. It takes the end-to-end transmission delay requirement for different subscription services into consideration, with an aim to reduce unexpected delays to the time-critical message exchange services. DRD-LB follows the same load management procedure specified in Section 2.4, which involves the processes of detecting the load states of brokers (load detection), distributing the load among available brokers (load distribution), analysing the load influence for each subscription service (load analysis), and migrating loads from overloaded brokers to the ones with less loads (offloading). The following is an overview of each process in DRD-LB.

- *Load detection* detects the load states of brokers. The load state of a broker is determined via periodically measuring the predefined load metrics of the broker and comparing them to corresponding thresholds. The details of the load detection design can be found in Section 4.4.1.
- *Load distribution* works when a subscriber registers with an HMA. HMA then uses an algorithm to select an edge broker and allocates it to the subscriber. This load distribution process aims to prevent a broker from becoming overloaded to avoid unnecessary load balancing by optimising the distribution of subscribers. The details of the load distribution design can be found in Section 4.4.2.

- *Load analysis* is part of the offloading process. It is the process of analysing the load influence of individual publishers and subscribers, e.g., it analyses the influence of each subscriber for each load metric specified for each edge broker. In addition, it generates a prioritised offloading list for each load metric when an overload is detected. The details of the load analysis design are presented in Section 4.4.3.
- *Offloading* happens when a broker becomes overloaded. It is accomplished by three steps: locating load-accepting broker(s), selecting clients to offload, and notifying the selected clients to migrate from the overloaded broker to load-accepting broker(s). Assuming that for the surplus resource case, brokers within each cluster can be added or removed automatically, on demand, the offloading process initially tends to be accomplished within the same cluster (named intra-cluster offloading). Only if all the available resources within a cluster are fully used, does offloading between clusters, named inter-cluster offloading, take place. The details of the offloading design are presented in Section 4.4.4.

In DRD-LB, it is further assumed that all the subscriptions belong to the same topic have the same delay requirement. In addition, in order to satisfy the end-to-end delay requirements specified by subscribers, subscribers are assumed to have enough network bandwidth capacity to receive all the matched publication messages. Thus, the corresponding queues in edge broker will be built up only when the network bandwidth of the broker is already fully used up.

4.4.1 Load Detection

Broker overload needs to be defined. Here, an Apache Qpid broker (see Figure 2-1) is used as an example. It is assumed that when a broker meets any of the following conditions, the broker is overloaded.

1) When the messages published exceed the total available download bandwidth (or input bandwidth) capacity of the broker, the throughput of the broker reaches its limit.
2) When messages received exceed the total matching capacity that the broker provides, i.e., input queue is built up to temporarily to host the messages that have been received but not processed.

3) When messages sent to subscribers exceed the total available upload bandwidth (or output bandwidth) capacity of the broker, the message output queue is built up temporarily to host the messages for the matched subscribers.

In order to detect accurately the load state of a broker, the load metrics and corresponding thresholds are specified for the head broker and edge broker respectively.

4.4.1.1 Load Detection Metrics

The main tasks of a head broker are defined as follows. First, it routes messages from head brokers of neighbour clusters and from local publishers to edge brokers within the same cluster that serve any matched subscribers. Second, it routes messages from local publishers to head brokers of neighbour clusters that have set up related federation paths. A head broker is less likely to be overloaded for doing the matching work as no subscribers connect to it [7]. Therefore, the load state of a head broker is mainly affected by the network bandwidth utilisation. An edge broker does much more matching work as it serves all the subscribers. Therefore, the load introduced by the matching process needs to be monitored.

Load Metric	Description	Head Broker	Edge Broker
U_i	*Input bandwidth Utilisation*	$\sum_{t \in T} \lambda_t / C_i$	
U_o	*Output bandwidth Utilisation*	$\sum_{t \in T} \mu_t / C_o$	
U_m	*Matching Utilisation*	N/A	$\sum_{t \in T} \sigma_t / C_m$

Table 4-2 Load Metrics for Head Broker and Edge Broker

Table 4-3 lists the load metrics used for head broker and edge broker. U_i, U_o and U_m are the utilisations for the input bandwidth, output bandwidth and matching capacity respectively. In addition, λ_t is the message-received rate in bytes/s for topic *t*. μ_t is the message output rate in bytes/s for topic *t*. T is the topic set for which the broker serves. σ_T specifies the number of message being exchanged for on topic *t*. The capacity for input bandwidth, output bandwidth and matching are denoted as C_i, C_o, and C_m respectively. Among these parameters, λ_t, μ_t, and σ_t are measured during the

operation time, i.e., the load detector periodically retrieved the value of these three parameters from the monitored broker. C_m is computed before the broker through factorial experiments and is assumed static. C_i and C_o are periodically measured using the Ping command. The detection period can be set from seconds to minutes. A shorter load detection period leads to a better in-time load detection but also introduces more communication overhead to the system. In a TWS, it is required that there is a fast system response to an overload. Therefore, detection occurs every second. In addition, in order to avoid unexpected jitter in the underlay network, which may cause a sudden increase and decrease in load, a weighted average method is introduced to get an average measurement to replace the instant measurements, i.e., $\sum_{i=1}^{N} w_i * v_i/N$. N is the window that defines how many measurements are used, v_i is the instant measurement for each time, and w_i is the weight set for each measurement. As TWS requires in-time load detection, N is set to two; the weights for the two measurements are set to 0.3 and 0.7 respectively. These values are modified to adapt to the network state, e.g., when the network becomes very unstable, the window size is set to a larger number such as 5 or 10.

4.4.1.2 Load State Determination

In this design, three different load states, LOW LOAD, HIGH LOAD, and OVERLOAD are introduced with two threshold values. The lower threshold indicates whether the broker has the ability to accept more subscriptions while the higher one is used to determine whether the broker is overloaded or not. This design is based upon [7]. LOW LOAD means that the broker's load is low and able to accept extra load, e.g. more subscriptions, as a load acceptor. HIGH LOAD means that the broker has enough work and is no longer available to accept more subscriptions until it goes back to LOW LOAD. OVERLOAD means that the broker is overloaded and needs to transfer load to another broker(s) with a LOW LOAD.

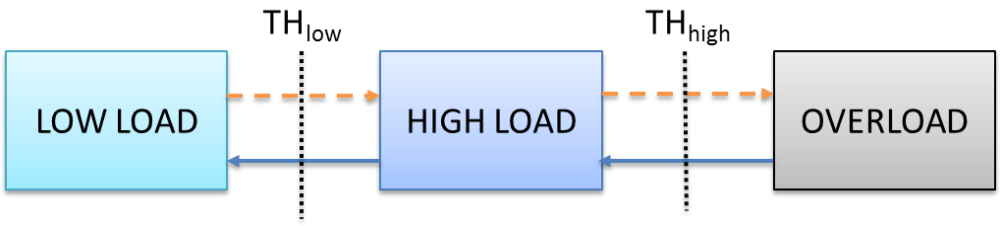

Figure 4-12 Load State Transfer

Figure 4-12 shows how the three load states relate to each other. The dashed arrow pointing right indicates an increase in load in the broker caused by an increase of publishing rate, or an increase of number of subscribers, etc. The solid arrow pointing left shows a load decrease inside the broker. Load balancing decreases the load of a broker with state OVERLOAD. In addition, a BUSY load state is used to label the brokers that are involved in the offloading process, which means a broker's load state is unstable at that moment.

The higher the value, the HIGH LOAD threshold (lower threshold) is set to, e.g., 99% Matching Utilisation, the more system resources that can be used. However, a broker may become overloaded before it can do any offloading. The magnitude of the difference between the lower and higher threshold controls the efficiency of load balancing and the level of the load imbalance between brokers. For example, a small difference, e.g., 1%, reduces the load imbalance between brokers but makes brokers more likely to enter OVERLOAD from HIGH LOAD, which may result in endless offloading cycles [7]. Thus, static thresholds are adopted and the value of higher and lower threshold are set as TH_{low} = 0.9 and TH_{high} = 0.95 respectively.

4.4.1.3 Load State Update

The LD periodically detects the load state of its broker. When the load state is detected, EMA sends a load state update to HMA, which contains the current load state of the broker and the estimated remaining capacity for each load metric. The remaining capacity is estimated as (TH_{high} – corresponding utilisation) * corresponding capacity. For instance, if the inBW utilisation U_i = 40%, input bandwidth of the broker is 5000KB/s, TH_{high} = 90%, the remaining capacity for inBW is estimated to be (0.9 – 0.4) * 5000 = 2500KB/s. This means the broker cannot accept message exchange above 2500KB/s according to the inBW Utilisation metric. EMA further compares the remaining capacity estimated for inBW utilisation, matching capacity utilisation, and outBW utilisation, the smallest one is used as the available remaining capacity and is reported to HMA.

When a HMA receives such information, it records the load state of the broker, the corresponding remaining capacity for each load metric and the estimated available remaining capacity. In addition, HMA prioritises the edge brokers according to the

available remaining capacity. The edge broker that has the maximum available remaining capacity has the highest priority to be allocated to a new subscriber client (the second load distribution principle defined in Section 4.4.2). In addition, when there are edge brokers that are in LOW LOAD state within the cluster, the load status of the cluster is set to be LOW LOAD. The total available remaining capacity of the cluster is also estimated for the edge and head brokers respectively. For the head broker, it is the available remaining capacity, while for edge brokers, it is estimated as the sum of the available remaining capacity for all the edge brokers in the cluster.

4.4.2 Load Distribution

In practice, it is necessary to avoid the overload problem by optimising the load distribution. With an H-E broker overlay, as all the publishers are assigned a head broker, load distribution focuses on how to assign brokers to subscribers.

In DRD-LB, the load distribution process is designed to address the following concerns. First, unnecessary network utilisation should be avoided in order to reduce the load of the cluster. Second, the broker that has the lowest chance to be overloaded should be assigned to a new subscriber. Third, regarding the delay requirements, the broker that has the lowest transmission delay to the subscriber should be assigned a new subscriber. Based upon these, the load distribution is designed according to the following principles:

1) Subscribers to the same topic are allocated to the same broker to avoid unnecessary network bandwidth utilisation, as the same message needs no longer to be routed to different edge brokers [7], which occupies N (the number of brokers that host subscribers with the same topic) times the bandwidth required.
2) Least-Capacity Utilisation Scheduling: if the cluster is deployed in a LAN environment, it is assumed the broker that has the most remaining capacity is considered with the least chance to be overloaded and thus, subscribers with new topics are allocated to those brokers whose capacity is least utilised.
3) Least-Distance Scheduling: if the cluster is deployed in a WAN environment, subscribers with new topics are allocated to brokers that are available to accept load (LOW Load state) and have the lowest transmission delay for message exchange with subscribers. In general, the brokers are the ones situated closest to a subscriber. This algorithm ensures that the subscriber has the lowest transmission

delay to the broker it initially connects to. It is used when brokers in a WAN environment are distributed across a large physical area, e.g., across different countries, covering the region being monitored (see Section 4.2).

The pseudo code of the load distribution principles is shown in Figure 4-13:

```
BEGIN
Let Mr = the registered message
Let Ctype = the type of the client retrieved from Mr, e.g., a sub or a pub
Let Mres = the response message sent to the subscriber with assigned broker
If Ctype == "sub"
Then
     Let t = the topic of the subscriber from Mr
     Let St = the set of topics that are subscribed by current subscribers
     If St contains t
     Then
          Let E = the edge broker information assigned for the existing subscribers
          subscribing to t
          Update Mres with E
          Send Mres back to subscriber
     Else
          Let Sb = the set of edge brokers in LOW LOAD state
          If the cluster is in LAN
          Then
               Let E = the edge broker that has the most remaining capacity
               Update Mres with E
               Send Mres to client registered
          Else
               Let E = the edge broker that is closest to the subscriber
               Update Mres with E
               Send Mres to client registered
          Endif
     Endif
Else
     Let H = the head broker of the cluster
     Update Mres with H
     Send Mres to client registered
Endif
END
```

Figure 4-13 Pseudo Code of Load Distribution

4.4.3 Load Analysis

Load analysis is achieved with the LA component. An LA aims to estimate and profile the load distribution for individual clients served by a broker, and to prioritise offloading clients according to the load metrics that cause the overload problem.

4.4.3.1 Load Distribution Estimation

Load distribution estimation aims to generate load distribution profiles. Here the load utilisation for each load metric for each topic and each client served by the monitored broker, e.g., publishers for head broker and subscribers for edge broker, are estimated and recorded.

Term	Description	Expression
U_i^t	inBW Utilisation for topic t	λ_t/C_i
U_o^t	outBW Utilisation for topic t	μ_t/C_o
U_m^t	Matching Utilisation for topic t	σ_t/C_m
U_i^p	inBW Utilisation for publisher p on topic t	$\frac{\lambda_p}{\lambda_t}U_i^t$ or λ_p/C_i
U_o^p	outBW Utilisation for publisher p on topic t	0
U_m^p	Matching Utilisation for publisher p on topic t	N/A
U_i^s	inBW Utilisation for subscriber s on topic t	0
U_o^s	outBW Utilisation for subscriber s on topic t	$\frac{\mu_s}{\mu_t}U_o^t$ or μ_s/C_o
U_m^s	Matching Utilisation for subscriber s on topic t	0

Table 4-3 Load Estimation Policy

Table 4-3 shows the load estimation policies. U_i^t, U_o^t, and U_m^t stand for the estimated utilisation for input bandwidth, output bandwidth, and matching capacity for topic t respectively. In addition, the inBW Utilisation, outBW Utilisation, and matching Utilisation for publisher p and subscriber s are estimated. The reason why U_o^p, U_i^s, and U_m^s are set to zero is that only when all the clients (both publisher and subscriber) belonging to topic t have been migrated to another broker, is the utilisation for that broker's load metric updated. As specified, matching utilisation is not considered in head broker and thus U_m^p is not measured. LA repeats the above processes for all the topics and the clients to generate load distribution profiles.

4.4.3.2 Offloading Clients Prioritization

In this design, clients for the same topic are treated as a (client) bundle in the offloading process, i.e., they are either migrated together to the load-accepting broker or kept together in the overloaded broker. Only if the load-accepting broker cannot accept any

bundle of clients, are these clients dealt with separately. Different clients are prioritised according to the load metric that causes the broker to be overloaded. This is used in the client selection process in the offloading phase, i.e., when LB selects client for offloading, the ones with higher priority are first considered. The principles to prioritise publishers and subscribers are described as follows, according to the load metrics in head broker and edge broker respectively.

4.4.3.2.1 Prioritise Publishers for Head Broker

In a head broker, according to where the matched subscribers are located, publishers of different topics can be categorised into four groups, P_r, P_l, $P_{r\text{-}l}$, and P_n, as shown in Table 4-4.

Type	Description
P_r	The topics published are only subscribed to by remote subscribers, which are served by other (or remote) clusters.
P_l	The topics published are only subscribed to by local subscribers, which are served by the same cluster as that for the publishers.
$P_{r\text{-}l}$	The topics published are subscribed to by both local and remote subscribers.
P_n	The topics published have no subscriptions.

Table 4-4 Publisher Classification

Table 4-5 shows the different offloading priorities for publishers in a head broker. If the head broker becomes overloaded due to a high inBW utilisation, the priority of all publishers is $P_n > P_r > P_{r\text{-}l} > P_l$. If the overload problem is caused by a high outBW utilisation, the priority relationship becomes $P_r > P_{r\text{-}l} > P_l > P_n$. The difference between the two is affected by the location of P_n, as migrating publishers with no subscribers cannot reduce the outBW utilisation but can only reduce the inBW utilisation.

Overload Metric	Priority
U_i	$P_n > P_r > P_{r\text{-}l} > P_l$
U_o	$P_r > P_{r\text{-}l} > P_l > P_n$

Table 4-5 Inter-Cluster Offloading Priorities for Publishers in Head Broker

In addition, for each type of publisher, the publishers for different topics have different priority depending on the delay requirements. Since the offloading process introduces

delays to the topic being exchanged, publishers with a more delay sensitive topic have a lower priority to be selected for offloading, i.e., they are less likely to exceed the delay requirements.

4.4.3.2.2 Prioritises Subscribers for Edge Broker

In each edge broker, similar to that in head broker, subscribers to different topics are categorised into groups S_r, $S_{r\text{-}l}$, S_l and S_n, as shown in Table 4-6.

Type	Description
S_r	The topics subscribed to are only published by remote publishers, which are served by other (or remote) clusters.
S_l	The topics subscribed to are only published by local publishers, which are served by the same cluster as that for the subscribers.
$S_{r\text{-}l}$	The topics subscribed are published by both local and remote publishers.
S_n	The topics subscribed have no publishers

Table 4-6 Subscriber Classification

In contrast to the publishers in a head broker, a subscriber may encounter two different types of offloading, intra-cluster and inter-cluster. Table 4-7 shows the different offloading priorities for subscribers in an edge broker for both intra-cluster offloading and inter-cluster offloading.

Overload Metric	Priority	
	Intra-Cluster	Inter-Cluster
U_i	$S_l = S_r = S_{r\text{-}l} > S_n$	$S_r > S_{r\text{-}l} > S_l > S_n$
U_o		
U_m		

Table 4-7 Intra-Cluster and Inter-Cluster Offloading Priorities for Subscribers in an Edge Broker

In intra-cluster offloading, the subscribers of type S_l, S_r and $S_{r\text{-}l}$ have the same priority to be selected. The reason is that, for these three types of subscribers, no matter what subscribed topic is published by local publishers or remote publishers, the messages are always routed from head broker to the edge broker, to which the subscribers connect. Thus, migrating any of these subscribers reduces the utilisation value for the

corresponding overload metric. In contrast, migrating subscribers that have no matched publisher does not reduce the utilisation of the overload metric. Thus, S_n has the lowest priority to be selected for offloading. For inter-cluster offloading, which is similar to the offloading of head brokers, the priority follows $S_r > S_{r\text{-}l} > S_l > S_n$ for all the metrics. For each type of subscriber, different topics are also prioritised depending on the delay requirements. A higher delay tolerant topic has a higher priority to be selected for offloading.

4.4.4 Offloading

When an overload is detected, offloading takes place. If a head broker becomes overloaded, offloading occurs amongst head brokers in different clusters by migrating publishers from an overloaded head broker to head brokers with less load. If an edge broker becomes overloaded, offloading first takes place within the same cluster, named intra-cluster offloading. Only if there is no available load-accepting broker in this cluster is inter-cluster offloading triggered, i.e., no brokers are in a LOW LOAD state, or the available load-accepting brokers have insufficient capacity to aid an overloaded broker to recover while in its OVERLOAD state. Both offloading processes follow a similar three-step offloading strategy, i.e., load-accepting broker locating, client selection, and client migration.

4.4.4.1 Intra-Cluster Offloading

The intra-cluster offloading only occurs amongst edge brokers. A three-step offloading strategy is described in the following sub-sections.

4.4.4.1.1 Locating the Load-Accepting Broker

As is specified in Section 4.4.1.3, the load state is periodically updated in EMAs and propagated to their HMA. Therefore, when a HMA receives an OVERLOAD state update, it updates the record for the corresponding edge broker and starts to locate the available edge broker within the cluster according to their load state. HMA selects edge brokers that have a LOW LOAD state and sends their corresponding IDs along with their remaining capacities for all the load metrics to the EMA of the overloaded broker. In case there is no available edge broker in the same cluster to offload to, HMA starts inter-cluster offloading. The detail of the inter-cluster offloading can be found in section

4.4.4.2. The pseudo code of the load accepting broker location process for HMA is shown in Figure 4-14:

```
BEGIN
Let L_E = the load state of an edge broker E
Let S = an empty list
If L_E == OVERLOAD
Then
    For each edge broker i
        Let L_i = the load state of the edge broker i
        If L_i == LOW LOAD
        Then
            Add i to S
        Endif
    End Loop
Endif
If S is NOT empty
Then
    Let M_can = the offload response message that contains the candidate load-accepting
    brokers
    For each edge broker j in S
        Let C_j = the remaining capacity of j
        Add C_j to M_res
    End loop
    Send M_res to E
Else
    Inter-cluster offloading is triggered
Endif
END
```

Figure 4-14 Pseudo Code for Locating the Load-Accepting Broker

4.4.4.1.2 Client Selection

Based on the results of step 1, the EMA of an overloaded broker prioritises the candidate brokers based on the remaining capacity of the load metrics of the brokers, i.e., the broker with the higher remaining capacity has a higher priority to accept the load. In addition, from the prioritised client list generated by the LA during the load analysis process, the EMA of the overloaded broker retrieves the load influence of each subscriber on each metric profiled by LA and estimates the load increase for the load-accepting broker for each load metric. For example, for U_i, the influence is estimated as the ratio of *input rate of the client* to *input bandwidth of the load-accepting broker*, which means that if clients are migrated to the load-accepting broker, U_i of the load-accepting broker will be increased by this amount. For the case that a client does not overload the load-accepting broker, it is selected and put in an offloading list. The selection process continues until the estimated load state of the overloaded broker is not OVERLOAD any more. The offloading list is then sent to the HMA. The HMA notifies

the EMAs of the selected edge brokers to be in a load-balancing phase and updates their states to "BUSY". Note that, for the case that there is not enough capacity left in the cluster for the load shifting, the EMA of the overloaded edge broker sends a request to HMA for inter-cluster offloading. The pseudo-code for this client selection process of an EMA is described in Figure 4-15.

```
BEGIN
Let E = the edge broker under monitored
Let M_can = the message that contains the information of candidate load-accepting
brokers
Let C_req = the required capacity for offloading
Let S_can = a set of candidate edge brokers retrieved from M_res
Let S_sub = an empty list that is used to record the information of selected subscribers
Let F = false
Sort S_can according to the remaining capacity from max to min
For each edge i in S_can
    Let C_i = the remaining capacity of i
    Let S_T = the topic list in the load distribution profile provided by LA
    For each topic t in S_T
        Let L_t = the load influence to i
        If C_i >= L_t
        Then
            C_i = C_i - L_t
            C_req = C_req - L_t
            Let Sub_t = the list of subscribers subscribing on topic t
            Add Sub_t to S_sub
        Endif
        If C_req <= 0
        Then
            F = true
            Break loop
        Endif
    End loop
    If F == true
    Then
        Break loop
    Endif
End loop
If F == true
Then
    Let M_sub = the message that contains the information selected subscribers,
    retrieved from S_sub
    Send M_sub to HMA
Else
    EMA sends request for Inter-Cluster Offloading
Endif
End
```

Figure 4-15 Pseudo Code of Client Selection

4.4.4.1.3 Client Migration

HMA sends migration notifications to all the clients that are in the offloading list, asking them to set up a connection to the load-accepting broker(s). All the clients then

set up connection(s) to the load-accepting broker(s) and drop the connection to the offloading broker, except for subscribers that have messages waiting in the queue in the overloaded broker. In this case, the subscribers drop the connections only when all the messages waiting in the overloaded broker are received. In addition, a message is sent by each client to HMA to confirm the completion of the migration process. The HMA counts the number of clients that have completed the migration away from the overloaded broker. There is also a default timeout for the migration so that the load-balancing phase can stop even if some clients close the connection during the migration. When all the clients complete the migration or the waiting time has timed out, the HMA notifies all the EMAs involved in the offloading phase that the offloading is complete and updates the states of edge brokers from "BUSY" to the actual load states, e.g., LOW LOAD and HIGH LOAD.

4.4.4.2 Inter-Cluster Offloading

Inter-cluster offloading is initiated by HMA when a head broker becomes overloaded or HMA receives a request from EMA of the overloaded edge broker, as intra-cluster offloading cannot resolve the problem of overloaded edge brokers by itself. The general process of inter-cluster offloading is roughly the same as that for intra-cluster offloading. The following sections give a description of these three steps.

4.4.4.2.1 Load-Accepting Broker Locating

When EMA initialises an inter-cluster offloading request or the head broker becomes overloaded, the HMA checks the recorded load state for neighbour clusters and locates the ones for which the load state is LOW LOAD. HMA sends an offloading request to the neighbour clusters to ask for the latest information about the remaining capacity. HMAs of neighbour clusters then check their local load states and report these to the HMA to initialise an offloading request. These HMAs state their remaining capacity for all the load metrics in their reports.

4.4.4.2.2 Client Selection

If the inter-cluster offloading is initialised by an EMA, HMA sends the remaining capacity for all the load metrics of neighbour clusters to EMA. EMA follows the same step in 4.4.4.1.2 to select clients from the prioritised subscriber list and to put the

selected ones in an offloading list with the corresponding load-accepting cluster. The offloading list is sent to the HMA and HMA forwards this to the corresponding HMA of the load-accepting broker. HMA of the load-accepting broker then changes the load state of the cluster and the edge brokers required to accept the load to "BUSY" state.

If the inter-cluster offloading is initialised by an HMA, HMA follows a similar strategy as that specified for the intra-cluster offloading, selects publishers from the prioritised publisher list and puts them in an offloading list. The offloading list is then sent to the HMAs of the selected load-accepting clusters. The load state of all the load-accepting clusters are then updated to "BUSY".

4.4.4.2.3 Client Migration

When the selection process is complete, the HMA of the overloaded cluster sends notifications to the client with the HMA information of the load-accepting cluster. It also sends a notification to the HMA of the selected cluster with the client information. When clients register with a new HMA, the HMA checks the client information and allocates it to a selected broker. When all the clients in the offloading list have been migrated or the timeout set for the offloading is reached, HMAs of both offloading cluster and load-accepting brokers update the load state of the cluster to its actually measured load state.

4.5 Validation

In this thesis, simulation-based experiments are undertaken to evaluate DRD-LB method for the surplus resource case by comparing it with PEER-LB. As specified in section 4.4.4, the offloading process for intra-cluster and inter-cluster are similar, thus, intra-cluster offloading is used as an example. In the following sub-sections, the experiment configuration and setup, the hypotheses, and the measurements are described. It is shown that DRD-LB outperforms the PEER-LB, whilst introducing less unexpected delay to subscription services. Therefore, DRD-LB seems much more suitable to be applied to time critical systems, such as for a TWS.

4.5.1 Experiment Configuration

4.5.1.1 Simulations and Assumptions

To compare DRD-LB with PEER-LB, the experiments focus on the overload problem and are designed using the following assumptions.

1. To simulate an intra-cluster overload problem, all the brokers involved are allocated within the same LAN and therefore the bandwidth is assumed constant during the experiments.
2. The matching capacity of each broker in each experiment is predefined and is assumed constant during the experiments.
3. The average publishing rate for each publisher is predefined and is constant during the experiments. A burst is simulated by increasing the publishing rate.
4. Published messages are generated based upon real sensor data and tsunami warning messages.
5. The transmission delay between publishers to brokers, between brokers, and between brokers to subscribers is pre-defined and is assumed constant for the surplus resource case.
6. Each subscriber has enough bandwidth to receive messages from a broker, and therefore this omits the slow subscriber problem.
7. Each subscriber has a pre-defined maximum transmission delay requirement that is assumed constant during the experiments.
8. Subscribers for the same topic have the same delay requirement while subscribers for different topics may have different delay requirements.

4.5.1.2 Experiments Setup

In this thesis, an intra-cluster offloading is chosen as an example. The setup used for the experiments involves four edge brokers (B0, B1, B2, and B3) connected to one head broker (Bh) to form a star topology. This experiment setup is used by the PEER framework [7]. Figure 4-16 shows an example of the broker overlay with four sets of matched publishers and subscribers. This intra-cluster simulates the broker deployment in a data centre.

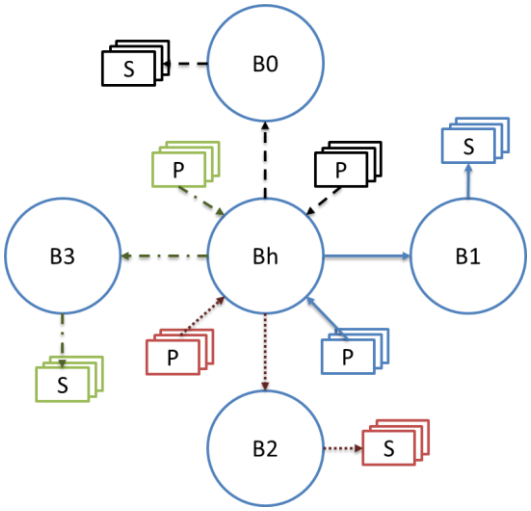

Figure 4-16 Broker Overlay Setup

Each broker has a predefined bandwidth and matching capacity. The bandwidth between all brokers is assumed constant at 1000Mb/s. The matching capacity for edge broker is randomly generated varying between 20000 to 30000 msg/s in each experiment. Based upon [7], the head broker does less matching work than edge brokers that serve subscribers. It is assumed that the head broker can process millions of messages each second in the experiments. Table 4-8 gives an example specification used for brokers in one experiment.

Broker ID	**Specifications**	
	Matching Capacity (msg/s)	***Bandwidth Capacity (Mbps)***
B_h	1,000,000	1000
B_0	27570	1000
B_1	25067	1000
B_2	24725	1000
B_3	22303	1000

Table 4-8 Broker Capacity Specification for an Experiment

The client information is designed as follows. In each experiment, one hundred topics tend to cover the range of topics used in TWS. These topics including topics for sensor data, workflow service, warning messages, user-generated data, and social network data such as Twitter. For each topic, the corresponding number of publishers, number of subscribers, publishing rate, and delay requirements are randomly generated. For example, delay requirements for different topics range from 1s to 50s. The size of each message is determined according to the topic it belongs to, e.g., for messages that contain sensor data, the size ranges from 200 bytes to 1 kilobyte.

Each experiment consists of four phases. First, clients of each topic are registered to the system each second and are allocated to a broker based upon the load distribution process (Section 4.4.2). This phase lasts until all clients are added to the system, i.e., 1s to 100s. Second, when publishers and subscribers operate normally, i.e., there is no reduction in bandwidth, nor any burst of message exchange. This phase lasts for 200s in each experiment, i.e., 101s to 300s. Third, the publishing rate for a range of topics (randomly chosen ranging from 10 to 15) is increased to 1.5 of its normal speed at 301s. This aims to simulate a tsunami scenario, so that the data acquisition rate from sensors and the corresponding workflow services are increased. After a few seconds, e.g., 100s, a burst for another range of publishing topics is simulated by increasing the rate to 1.5 times of their normal rate. When any of the brokers is determined to be overloaded, offloading is triggered. The time cost for offloading is recorded. Fourth, after another duration, e.g., at 900s, the message dissemination rate and data generation rate for all the topics returns to their normal rate and the system returns to equilibrium.

4.5.2 Hypothesis to Evaluate

To evaluate the performance of DRD-LB with comparison to the baseline method PEER-LB, the following hypothesis is proposed.

Hypothesis 1 (H1): *Compared to the state of the art load balancing method proposed for PEER framework, DRD-LB for ePEER introduces fewer unexpected delays to delay sensitive subscriptions.*

In contrast to conventional load balancing methods, during the load analysis phase, DRD-LB takes the delay requirements for different subscription services into consideration. Hence, a subscriber that is delay sensitive has a lower priority to be selected for migration when a broker to which it is attached, becomes overloaded. Therefore, delays caused by migration to time-critical services are reduced.

4.5.3 DRD-LB Performance Metrics

To evaluate the hypothesis, the following metrics are determined during the experiments.

1) Communication Overhead

Load detection utilises system processing and communication capacity to retrieve and analyse the load metrics to determine the load state. The communication overhead is expressed as the ratio of message exchange rate of load detection messages to overall message exchange rate. For example, if one load detection message is sent every second to retrieve the load status of the broker, and the messages being exchanged through the broker for each second are 50000, the communication overhead is then computed as 1/50000.

2) Number of Offloads

Since offloading requires system resources to select clients and notify them for migration, it introduces processing and communication overhead to the system. A more efficient load balancing solution requires less offloading to balance the load among brokers. Thus, this parameter represents the efficiency of the offloading algorithm for different load balancing methods.

3) Offloading Delay

This metric is used to represent the unexpected delays introduced to different subscription services. For each subscriber, the offloading delay, *OD*, is measured as the ratio of *unexpected delay* to *delay constraint*. For each offload event, the sum of the individual offloading delays is computed. A lower value indicates that the offloading method is less likely to violate the delay requirements.

Table 4-9 gives a summary of how to measure the above performance metrics in each experiment.

Metrics	Measurement
Communication Overhead	Load Detection Rate / Matching Capacity
Number of Offloads	The number of offloads triggered to balance the broker when an overload is detected.
Offloading Delay	In each experiment, for each subscriber, the offloading delay is measured as the unexpected delay / delay constraint, i.e., $O_d(s) = T_{unexpected}/T_{SLA}$, where T_{SLA} is the delay requirement; and for each experiment, the overall offloading delay is $O_d = \sum_{s \in S_o} O_d(s)$, where S_o means the set of subscribers that are notified to offload. Thus, the overall average offloading delay for all the experiments is $\overline{O_d} = \sum O_d/N$, where N is the number of experiments carried out.

Table 4-9 Performance Metrics of DRD-LB

4.5.4 Validation Results

The experiments are repeated 500 times. As specified in Section 4.5.1, in each experiment, the configuration of the experiment varies, e.g., the publishing rate and the capacity of the brokers are randomly generated. In each experiment, the load metrics for each broker is measured periodically, i.e., the load detection period is set to one second. The lower threshold is set to 90% and the higher threshold is set to 95%. The threshold values vary with respect to the system resources and the requirements of the application scenario. In addition, to evaluate the hypothesis, the metrics specified in 4.5.2 are also measured.

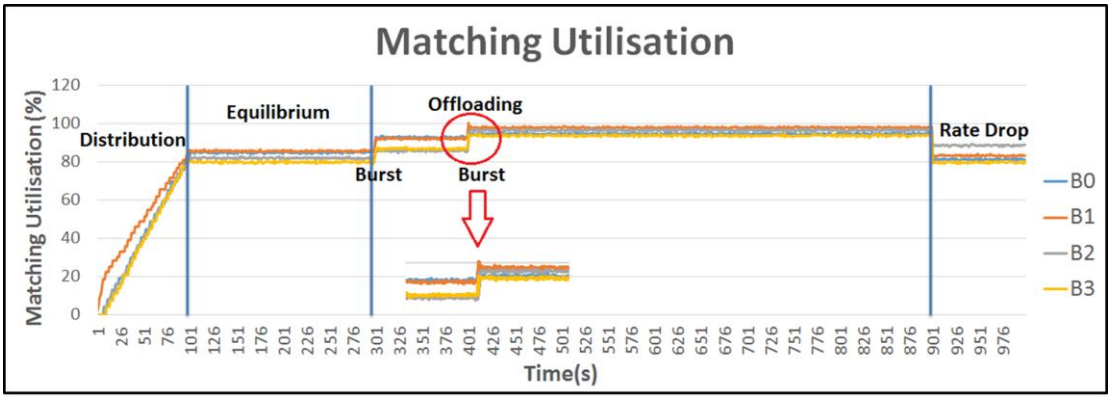

Figure 4-17 Matching Utilisation with DRD-LB

Figure 4-17 shows an example of capacity matching utilisation measured for one experiment with DRD-LB, in which two offloads are triggered after the 2nd burst. For each simulated burst (for the 500 experiments), the message rate increases randomly, some of these may require more than one offload to balance the load, e.g., 4 offloads to balance the broker. In addition, as one load detection message is sent each second to detect the load status of the broker, the communication overhead introduced by the load detection is measured as 1/matching capacity, e.g., for broker B0, the communication overhead is computed as 1/25887 in this experiment. The number of offloads triggered is two. The offloading delay for this experiment O_d is computed as 3.92%. After the 500 experiments, the offloading delay $\overline{O_d}$, which is measured as the average offloading delay for each experiment, is computed as (24.918/500)*100% = 4.98%. This means that on average, DRD-LB introduces a 5% offloading delay to all the message exchange services.

Similar experiments to these have been carried out for the baseline method PEER-LB. With PEER-LB, publishers connect to head brokers while subscribers connect to edge brokers. However, in contrast to the design of DRD-LB, subscribers are randomly assigned to an edge broker. PEER-LB measures the load difference between any edge brokers. Once the difference between two edge brokers exceeds a threshold, e.g., 10% [7], load balancing is triggered to balance the load between these two.

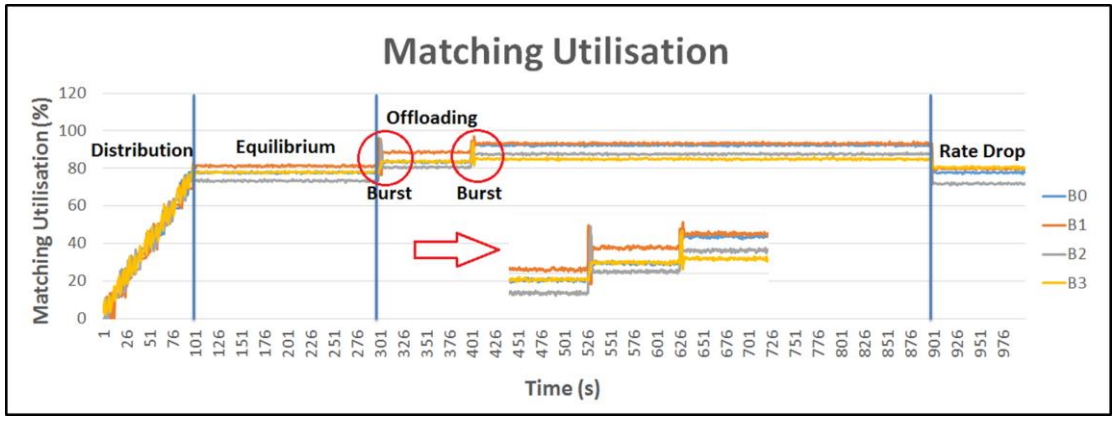

Figure 4-18 Matching Utilisation with PEER-LB

Figure 4-18 illustrates an example of the capacity matching utilisation for one of the 500 experiments using the PEER-LB as a baseline. In this experiment, during the load distribution phase, a significant amount of offloading is triggered to balance the load between brokers. After the first burst, since the rate increase is randomly chosen, the load difference between brokers may exceed its threshold, and may require offloading. This works similarly when the second burst is simulated. As specified in Table 4-9, the communication overhead is measured as the ratio of the detection rate to the matching capacity, e.g., for B0, the communication overhead by load detection is computed as 1/24031 in this experiment. During the 500 experiments, the number of offloads triggered is 220 including the distribution phase, and 18 excluding the distribution phase. The offloading delay for this experiment is measured as 2319% (including the load distribution phase) and 122% (excluding the load distribution phase). The performance is similar to other experiments, and after 500 experiments, the average offloading delay is then computed as 1894% and 274% with and without the load distribution process respectively.

Figure 4-19 illustrates the number of offloads difference between PEER-LB and DRD-LB. Note that the result presented for PEER-LB excludes the offloading that happens during the load distribution phase. In the diagram, a positive value indicates that PEER-LB adopts more offloads while a negative value indicates that DRD-LB adopts more offloads. It is clearly shown that for the 500 experiments, only 7 of them, i.e., 1.4%, does the DRD-LB require more offloads; in 5 of them, i.e., 1%, DRD-LB and PEER-LB adopt the same number of offloads. In the remaining 97.6% of experiments, DRD-LB requires less offloads to balance the load between brokers.

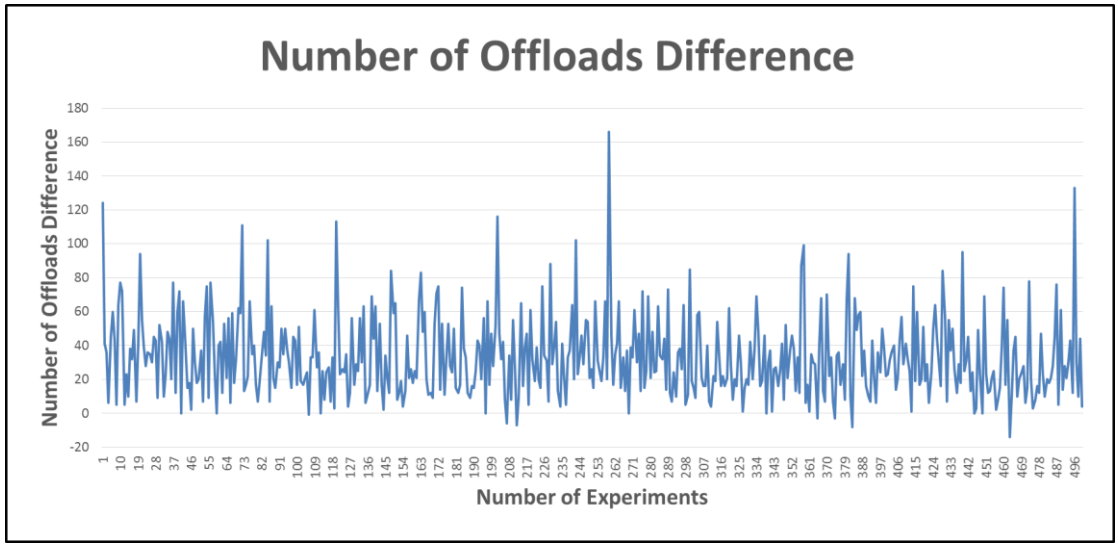

Figure 4-19 Number of Offloads Difference between PEER-LB and DRD-LB

Figure 4-20 illustrates the offloading delay difference between PEER-LB and DRD-LB. Similarly, a positive value indicates that PEER-LB introduces more offloading delays, while a negative value means DRD-LB introduces more offloading delays. As is shown in Figure 4-20, only in three of the 500 experiments, DRD-LB introduces more offloading delays. The offloading delay is influenced by the delay requirements of the subscriber selected, as described in Table 4-9. Thus, since DRD-LB is aware of the delay requirements of different subscribers, even if in some cases DRD-LB adopts more offloads, less offloading delay is introduced to the system.

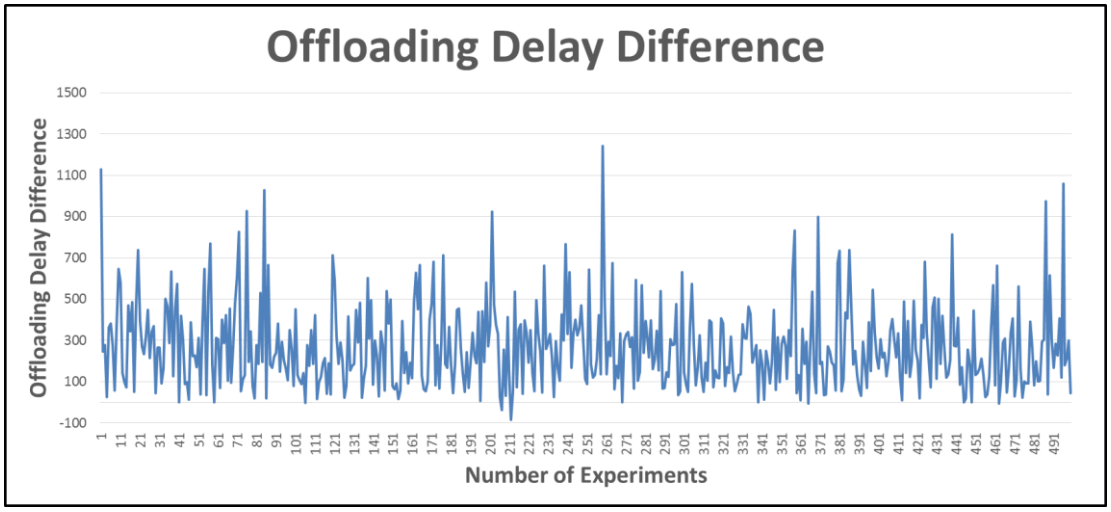

Figure 4-20 Offloading Delays Difference between PEER-LB and DRD-LB

The average number of offloads and average offloading delays between PEER-LB and DRD-LB are also computed. The results are illustrated in Figure 4-21 and Figure 4-22.

Note that the result presented for PEER-LB excludes the offloading that happens during the load distribution phase. According to the results, it is clearly shown that DRD-LB outperforms PEER-LB while requiring less offloading and introducing less delay for time-critical subscription messages.

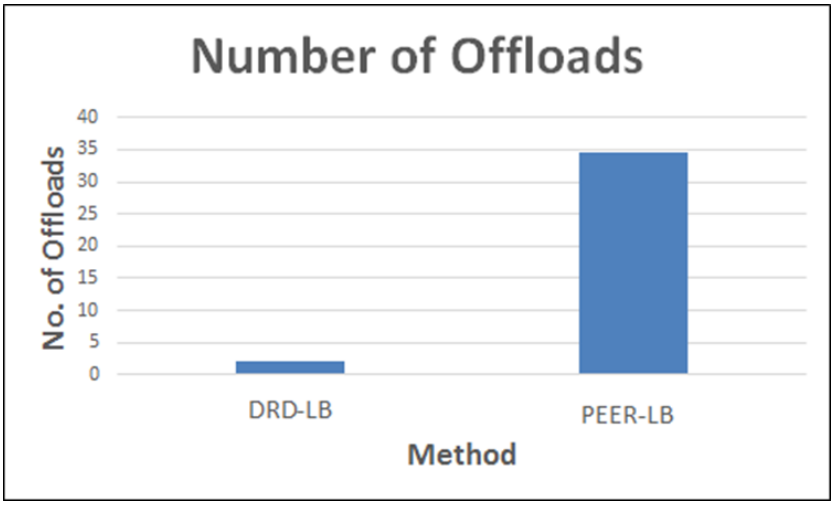

Figure 4-21 Average Number of Offloads between DRD-LB and PEER-LB

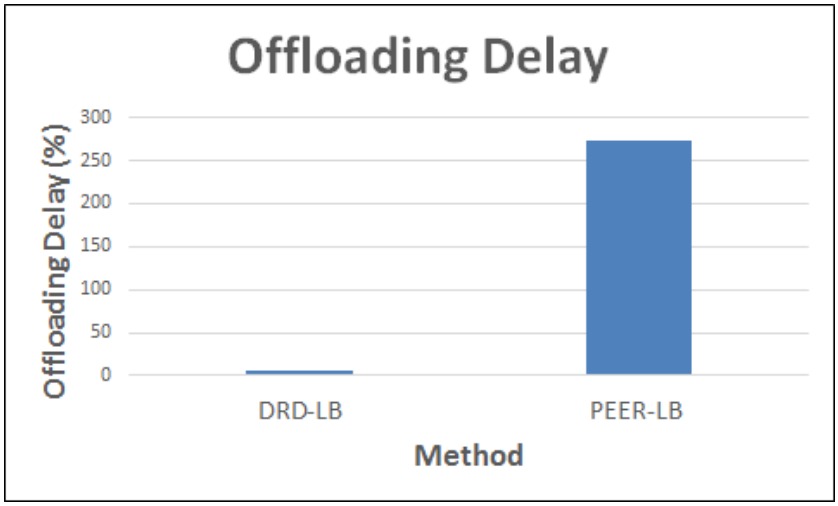

Figure 4-22 Average Offloading Delay between DRD-LB and PEER-LB

To prove Hypothesis 1, a null hypothesis (H_{null}) and its alternative hypothesis (H_{alt}) are introduced. H_{null} assumes that DRD-LB for ePEER introduces a similar offloading delay as PEER-LB, while H_{alt} assumes that DRD-LB for ePEER introduces less offloading delay than PEER-LB. To check whether H_{null} is true, the offloading delays of all the 500 experiments for DRD-LB for ePEER and PEER-LB are retrieved to form two samples (S_{ePEER} and S_{PEER}). To assess whether the offloading delay between DRD-LB and PEER-LB is significant, a Wilcoxon signed-rank (non-parametric statistical

hypothesis) test is used to compare these two samples. The following assumptions need to be satisfied for the test to be valid. First, the data in two different samples are paired and come from the same population. Second, each pair is chosen randomly and independently. Third, the data is measured at least on an ordinal scale. In this case, both S_{ePEER} and S_{PEER} contain 500 offloading delays measured for each experiment. All the data in a sample are indexed, e.g., the offloading delays in sample S_{ePEER} are marked from $S_{ePEER}(1)$ to $S_{ePEER}(500)$. Each data in one sample is paired with data in another sample with the same index, e.g., $S_{ePEER}(1)$ is paired with $S_{PEER}(1)$. After applying these two samples as input parameters into a Wilcoxon test, which is achieved using R's "wilcox.test()" function, a p value that reflects the chance that H_{null} happens is computed. In this case, the p value is computed as $p<2.2e-16$, which is close to 0. This means that H_{null} rarely happens and the alternative hypothesis H_{alt} is shown to be valid, i.e., DRD-LB for ePEER introduces less offloading delays than PEER-LB. Hence, Hypothesis 1 is true.

To demonstrate that the load analysis methods adopted in DRD-LB can also improve the performance of PEER-LB, 500 more experiments are carried out. In these experiments, the load analysis methods used in DRD-LB are applied into PEER-LB, i.e., the subscribers are prioritised following the same principles specified in Section 4.4.3.2. This new LB method is denoted as PEER-DRD. The same performance metrics are measured for each experiment. Considering these three performance metrics, since the load distribution and load detection algorithms adopted in PEER-LB have not been changed, the communication overhead metric and the number of offloads metric are not improved, the offloading delay metric improves markedly. The average offloading delay metric for these 500 experiments is measured as 1022% and 137%, with and without the load distribution process, respectively. Compared to 1894% and 274% for the experiment with the pure PEER-LB method, the average offloading delay is reduced by a third. Figure 4-23 shows the comparison of offloading delay between original PEER-LB, and PEER-DRD that adopts the load analysis method for DRD-LB. The result clearly shows that by applying the load analysis method adopted in DRD-LB, PEER-DRD introduce less delays to time-critical messaging services.

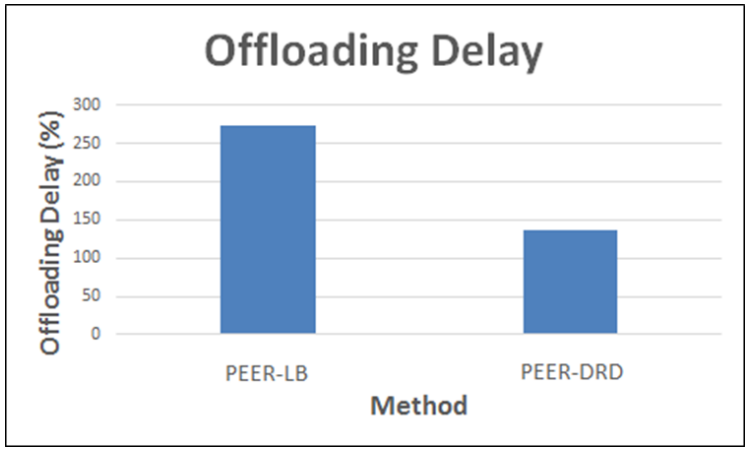

Figure 4-23 Comparison of Offloading Delay between PEER-LB and PEER-DRD

4.6 Summary

This chapter describes a load management framework, ePEER, which extends the existing PEER framework. This describes the design of the broker overlay, the management agent, and the load-balancing life cycle for the surplus resource case. The H-E broker overlay is constructed based upon management agents, which communicate using pub-sub type standardised AMQP message exchange. For a TWS application, MAs are used for load management of the sensor data, workflow services, and warning message exchange. In such a H-E broker overlay, when some brokers become overloaded due to information bursts and when there are surplus system resources, a delay requirement driven load balancing method, DRD-LB, is invoked to balance the load between overloaded brokers and lightly loaded ones to minimise the delay for different topics. A comparison between DRD-LB and the baseline load-balancing method proposed in PEER, named PEER-LB, is presented based upon realistic TWS simulation-based experiments. Each experiment is repeated 500 times and the average performance value is obtained. The validation results show that for the time-critical systems, such as TWS, DRD-LB outperforms the baseline PEER-LB method to ensure that fewer delays are introduced to time-critical messaging services. In the next chapter, the load management method of ePEER for the limited resource case is described.

5 FEEDBACK DRIVEN CONGESTION CONTROL FOR LIMITTED RESOURCE CASE

5.1 Overview

In Chapter 4, a load-balancing method of ePEER framework, DRD-LB, which balances the load amongst brokers in an H-E overlay has been discussed and validated. However, this load balancing solution cannot solve the broker overload problem for the situation where a system's resource limits are reached and cannot be scaled up. Such a situation occurs when a physical environment crisis, e.g., an earthquake happens, as it may disrupt the underlying network infrastructure and hence decrease the available system resources (see Section 1.1 for an example). Such damage severely reduces the capacity of links from publishers to brokers, between federated brokers, and from brokers to subscribers. In this case, load balancing through migrating subscribers or publishers from an overloaded broker to another (less loaded) broker, may not resolve the broker overload problem. The available system resource limits are not only over-stretched by broken links but also by an increased rate of exchange of information from human sources and sensors as publishers at the onset of crises. Published information from some of these publishers may have no matched subscribers, and may be repetitive or of little value; matched subscribers may not find such content exchange that important. These overactive publishers introduce an unnecessary load on brokers and on the broker links. This may result in an over-demand to utilise the processing and communication resources of the PSMOM system. To supplement the DRD-LB load balancing method, a congestion control model, the Feedback Driven Congestion Control (FDCC) model, is introduced.

5.2 FDCC Design

FDCC is designed to limit the message-publishing rate of overactive publishers by applying a filter at the publisher. The publishing rate limit is determined based upon the utility of the messages to subscribers, the utility of publishers, and the utility of

topics computed by the UA and SA components in HMA. HMA sends notifications to publishers when it is necessary to limit the publishing rates. In contrast to the existing congestion control methods discussed in Section 3.5, FDCC is designed for the situation when the system resources are already fully used. In addition, FDCC limits the publishing rate of overactive publishers based upon the utility of the published messages and the importance of the publishing topics to maintain the QoE for subscribers. These make FDCC a better choice to manage broker load for PSMOMs in a TWS when the system resources are already fully used.

A message controller component that sends messages to a broker is introduced. It is used to convert the raw data, e.g., sensor data, into MOM messages. Similarly, a message extractor is used by subscribers to receive messages and to extract their message content. In practice, the data sources that provide the raw data and message controllers can be deployed in different types of system and connected via wireless or wired connection. This also applies to the message extractor and data consumer. Figure 5-1 shows an example of H-E broker overlay with message controller and message extractor. In this example, one head broker and three edge brokers with their corresponding management agents form an H-E broker overlay. HMA can send management messages to publishers and subscribers to alter the behaviour of the associated clients through the OM component specified in Section 4.3.1.

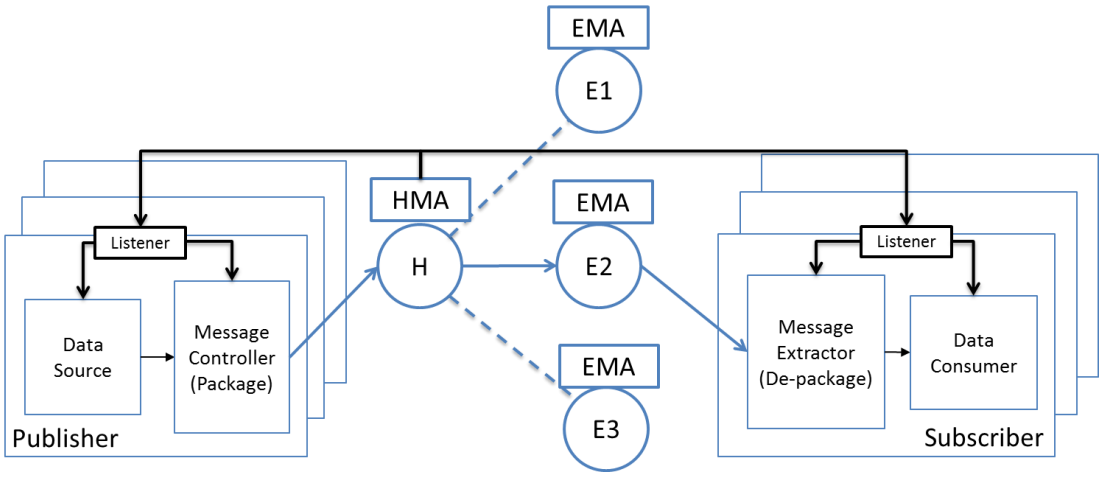

Figure 5-1 H-E Broker Overlay with Message Controller and Message Extractor

Messages published to the head broker are generated by the message controller in the publisher (see Figure 5-1). This generates PSMOM messages using the raw data from the data source, e.g., database or user inputs. The message controller acts as a filter

between the data source and the H-E broker overlay. Alternatively, when a message is received by a subscriber, the message extractor retrieves the message content and passes the content information to the data consumer. Figure 5-2 shows an example of how to apply a message controller at a publisher that uses a filter to limit the rate of publishing to facilitate the limited resource case.

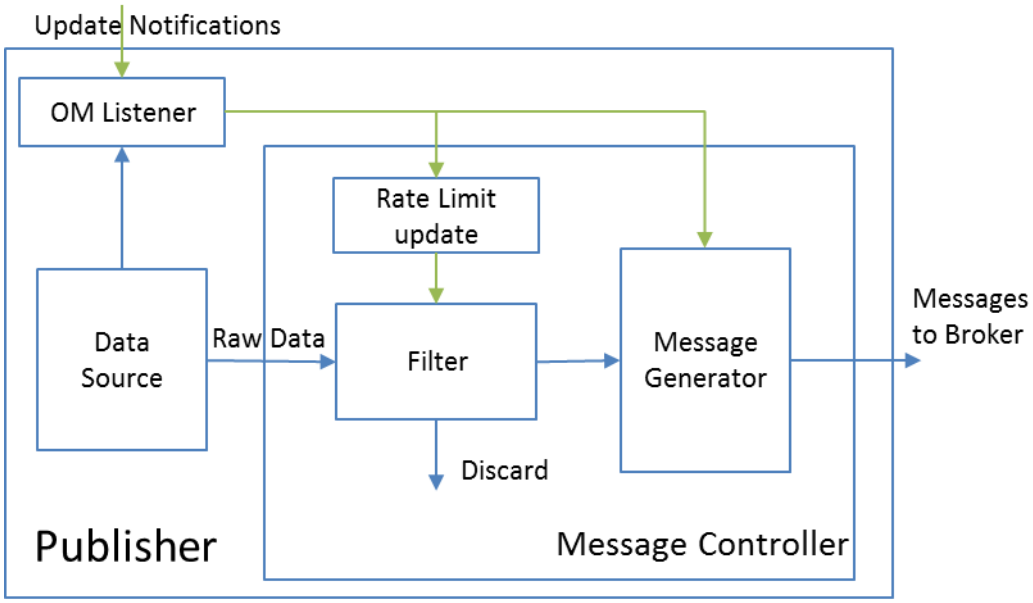

Figure 5-2 Pre-Filter Use in a Publisher

As shown in Figure 5-2, a rate limit filter sits between the data source and message generator. The reason to put the filter before the message generator is to save the system resources by avoiding generating messages that may be discarded, i.e., messages are generated using raw data from data sources only when the data satisfies the filtering criteria. In the following sub-sections, the details of the FDCC design are described. Note that it is assumed that the PSMOM system does not queue publishers' messages that have no matched subscribers.

5.2.1 Rate Limit Filter

A rate limit filter limits the publishing rate of a publisher. It receives the data for which the publishing topic has matched subscribers. The filter then compares the current publishing rate with the maximum allowed publishing rate set by HMA. If the current rate is less than the rate limit, the data is forwarded to the message generator; otherwise, the data is discarded.

The maximum allowed publishing rate is determined based upon the utility of publisher, denoted as U_p, which is computed by HMA based upon the feedback of the utilities of the published messages provided by the matched subscribers. The following assumptions are made in the design. First, each subscriber only subscribes to one topic. Second, each subscriber has the knowledge of where the received message is from, e.g., from which publisher. This is achieved by retrieving the *pubID* that is added by the publisher and stored as a property of each received message. Third, all the subscribers compute the utility of messages for each publisher that are of most interest to them and update the values to HMA every unit time, e.g., every second. A high feedback frequency provides a more accurate determination of the utility of the publisher but there is a trade-off in that this adds a communication and processing overhead as each feedback consumes system resources to receive and process messages. In this design, the feedback frequency is set to 1Hz, which means in each second, there is only one feedback. The following gives more detail of how to limit the rate of a publisher with respect to the utility of the published messages.

Based upon these assumptions, a subscriber *s* assesses the utility of each received message. Specifically for subscriber *s*, and message *i* from publisher *p*, the utility of this message, denoted as $U_m(s,i,p)$, is decided by the subscriber. However, to reduce the communication load in exchanging the utility values, a subscriber does not send each U_m to the HMA. Instead, the subscriber computes the average value of the utility of messages from the same publisher and only sends this average value to HMA each second. That is, if it is assumed that there are $N_m(s,p)$ messages received by subscriber *s* from publisher *p*, in a second, the average utility from subscriber *s*, for publisher *p*, denoted as $\overline{U_m(s,p)}$ is computed using Equation (1).

$$\overline{U_m(s,p)} = \frac{\sum_{i=1}^{N_m(s,p)} U_m(s,i.p)}{N_m(s,p)} \quad (1)$$

When HMA receives the average utility value of each publisher from each matched subscriber, it computes the utility of the publisher. Equation (2) defines the utility of publisher for publisher *p*, $U_p(p)$, where: $W(s)$ is the weight assigned to the subscriber *s*; $S(p)$ is the set of subscribers that provide the average utility value and subscribe on the matched topics of publisher *p*; $|S(p)|$ is the number of the matched subscribers of publisher *p* that provide feedback.

$$U_p(p) = \frac{\sum_{s \in S(p)} W(s)\overline{U_m(s,p)}}{|S(p)|} \quad (2)$$

For each publisher *p*, the publisher's rate is limited based upon the utility value. The range of U_p is normalised to be the range of U_m, i.e., [0, 1]. For a simple control mechanism, the range of U_p is divided into 5 regions, i.e., [0-0.2), [0.2-0.4), [0.4-0.6), [0.6-0.8), [0.8-1]. Here each region has a corresponding publishing rate limit that is defined according to the resources of the broker, e.g., 10 msg/s (messages per second), 200 msg/s, 500 msg/s, 900 msg/s, and 2000 msg/s. This means that, for example, if a publisher has an U_p value of less than 0.2, it is allocated a maximum publishing rate of 10 msg/s. When the utility of a publisher is changed, HMA updates the publishing rate limit by sending a notification to the corresponding publisher.

5.2.2 Topic Selection

When the system resources are fully utilised, to avoid broker performance degradation, HMA starts to select publishers to continue publishing messages according to the importance of each publishing topic. This selection process is called topic selection.

The first step of determining the importance of a topic is to determine the utility of a topic. This is computed by HMA based upon the utility of the publishers for the same topic. If $U_t(t)$ is used as the utility of the topic *t*, $P(t)$ as the set of publishers that publish messages under topic *t*, $U_p(p)$ as the utility of a publisher *p*, $U_t(t)$ is calculated using Equation (3), where $|P(t)|$ stands for the number of publishers in the set.

$$U_t(t) = \frac{\sum_{p \in P(t)} U_p(p)}{|P(t)|} \quad (3)$$

HMA then computes the importance of each topic *t*, denoted as $I_t(t)$, which is proportional to the utility of the topic, $U_t(t)$, the number of subscribers that subscribe to the topic, $S(t)$, and inversely proportional to the system resource (such as matching capacity) required, $R_t(t)$, as shown in Equation (4).

$$I_t(t) = \frac{U_t(t)S(t)}{R_t(t)} \quad (4)$$

Once the importance of each topic is computed, the topics are recorded in a topic list, *T*, in which the topics are ordered by its importance. When system resource limits are reached, HMA starts to select the publishers that can continue publishing, i.e., the topics that are considered more important. The constraint on the selection process is that the sum of the required bandwidths for the selected topics should not exceed the total outBW, i.e.,$outBW \geq \sum_{i \in T} reqBW_i$ where *T* is the topic list in a descending order by the importance. Whenever the required bandwidth for all the publishers publishing on a topic exceeds the remaining output bandwidth, not all publishers for that topic are selected to publish it as follows. To achieve this, the HMA starts to check the publisher lists. In those publisher lists, publishers for the same topic are ordered by their corresponding utility values in descending order. Publishers with higher utility values are selected such that the accumulated required bandwidth does not exceed the total outBW.

For the topics not selected to continue being published, a rate update notification is sent to all the publishers that publish messages for the topic. When a rate update notification is received by a publisher, indicating that a topic can no longer be published or be published at its full speed, the rate limit filter limits the publishing rate accordingly. For instance, if the publisher is notified to stop publishing during the limited resource case, the rate limit filter discards all the data sent to the filter, in order to provide more system resources for message exchange on more important topics.

5.3 Validation

In this thesis, simulation-based experiments are carried out to evaluate FDCC method for the limited resource case.

5.3.1 Experiments Configuration

5.3.1.1 Simulation and Assumptions

The experiments follow the same simulation and assumptions as that specified in 4.5.1.1 with the following additional ones.

1. The infrastructure damage caused by the crisis is simulated by reducing the bandwidth of all the brokers by half. These values are assumed to remain constant and cannot be scaled up.
2. Each subscriber is required to provide feedback to HMA each second. The feedback consists of the utility value measured by a utility function according to the interested content and the content being received.
3. Each publisher is designed with a rate limit filter, as specified in 5.2, which is under control of HMA to limit the publishing rate.

5.3.1.2 Experiment Setup

As specified, the limited resource case can happen when the physical network infrastructure has been damaged. To validate the proposed FDCC method, an intra-cluster H-E broker overlay that simulates the broker overlay in a Tsunami affected area is adopted. The setup used for the experiments involves four edge brokers (B0, B1, B2, and B3) connected to one head broker (Bh) to form a star topology. The setup is the same as that adopted for the experiments described in Section 4.5, i.e., the capacity and bandwidth of each broker is randomly generated ranging from 20000 msg/s to 30000 msg/s and 300Mbps to 500Mbps respectively. An example specification for the broker capacity is described in Table 5-1.

Broker ID	**Specifications**	
	Matching Capacity (msg/s)	***Bandwidth Capacity (Mbps)***
B_h	1,000,000	1000
B_0	26015	368
B_1	24112	448
B_2	22104	384
B_3	20912	464

Table 5-1 Broker Capacity Specification for an Experiment

The client information is designed as follows. In each experiment, 100 topics are used in a TWS for the affected area. This includes topics for warning message dissemination, evacuation advice from public agents, and human generated help and advice messages that are exchanged with people in an affected area. For each topic, the corresponding number of publishers, number of subscribers, publishing rate, and delay requirements are randomly generated. The size of each message is randomly generated ranging from 200 byte to 2 kilobyte. In each experiment, each subscriber has a random number of

keywords and each message published contains a set of keywords that are randomly generated. These are used to compare with the keywords specified for the corresponding subscriber to compute the utility of the publisher. The content of each message is retrieved from twitter with initial keyword "Tsunami" using Twitter Crawler [83]. It is assumed that 50% of the publishers are good publishers that publish information that are satisfied by the subscribers; 25% of the publishers are medium publishers that publish part of information required by the subscribers; the rest are bad publishers or overactive publishers that publish information that is of no use. In each experiment, subscribers provide one feedback message each second. Although providing thousands of feedback messages per second can improve the accuracy of the measurement, this volume of feedback introduces a larger communication and processing overhead to the system, as it has to deal with receiving and processing each feedback message.

Each experiment lasts 1000 seconds and consists of four phases. The first two phases are the same as that specified for the DRD-LB experiments. At the beginning of the third phase, e.g., at 301s, a disruption to the physical network is simulated by reducing the bandwidth of each broker to 0.1 to 0.4 of its original amount, e.g., if a broker is set with a bandwidth 1000 Mbps originally, it becomes 100 to 400 Mbps after this simulation. The amount reduced is randomly generated. After the simulated damage happens, the system resources become limited and FDCC is triggered when some brokers become overloaded. This phase lasts until 900s, when the final phase starts. In the final phase, the bandwidth of all the brokers recovers to the original amount and the publishers whose publishing rates were limited are notified, so that the system can recover to its original state.

5.3.2 Hypotheses to Evaluate

To evaluate the performance of FDCC, two further hypotheses are proposed.

Hypothesis 2 (H2)：*FDCC limits the publishing rate of overactive publishers, which reduces the load to brokers.*

FDCC measures the utility of publishers based upon the feedback from matched subscribers. According to this utility value, the publishing rates by publishers are

limited if the utility value is low. This reduces the message rate to brokers and therefore, reduces the load to brokers.

Hypothesis 3 (H3): *FDCC allows messages with a high utility value, i.e., messages with important information, to continue being published and therefore improves the QoE of subscribers.*

FDCC limits the publishing rate according to the utility of publishers and importance of topic. Therefore, when the resources are limited, only the publishers that publish messages for a relatively more important topic continue publishing. This design ensures that subscribers to have more chance to receive the more important messages rather than to receive the messages with little value. Thus, it is possible that the average QoE for subscribers is improved.

5.3.3 FDCC Performance Measurements

To measure the performance of FDCC, the following parameters are selected.

1) Average QoE for All the Subscribers

The average value of QoE of all the subscribers, denoted as $\overline{E_s}$, which is the output of a utility function, is used as the controlling parameter. A higher utility function value indicates a higher QoE for all the subscribers to the PSMOM system. If $E_s(s)$ is the QoE for subscriber *s*, $W(s)$ is the weight assigned to subscriber *s*, and *S* stands for the set of all the subscribers, $\overline{E_s}$ is computed using Equation (5), where $|S|$ stands for the number of subscribers in set *S*.

$$\overline{E_s} = \frac{\sum_{s \epsilon S} W(s) E_s(s)}{|S|} \tag{5}$$

For the QoE of subscriber s, $E_s(s)$, if $P(s)$ is used as a set of publishers that publish messages for the same topic subscribed to by subscriber *s*, $\overline{U_m(s,p)}$ is the average utility value for publisher *p* from subscriber *s*, $E_s(s)$ is computed using Equation (6), where $|P(s)|$ is the number of publishers in the set.

$$E_s(s) = \frac{\sum_{p \in P(s)} \overline{U_m(s,p)}}{|P(s)|} \quad (6)$$

Applying Equation (1) and (6) into (5), the average experience for all the subscribers is expressed as shown in Equation (7).

$$\overline{E_s} = \frac{\sum_{s \epsilon S} \frac{W(s) \sum_{p \in P(s)} \frac{\sum_{i=1}^{N_m(s,p)} U_m(s,i.p)}{N_m(s,p)}}{|P(s)|}}{|S|} \quad (7)$$

In practice, the utility value for each message is subjective to each subscriber. In this design, a simple model is built to generate the utility for individual messages. In this model, each subscriber *s* defines a set of interest keywords *NK(s)* that can be a sub-set of the keywords defined for the subscribed topic. These interest keywords are used to filter received message content to select which messages are of the most interest to subscribers and used to generate a utility value for each message. It does this by measuring how many keywords are matched in the content of each received message. The key interest key words can be tailored to each *s*. These keywords can be pre-set or specified dynamically. If *MK(s,i)* is the set of matched keywords for message *i*, in which the keywords is denoted as *NK(i)*, the utility for each message *i* published by publisher *p* and received by subscriber *s*, denoted as $U_m(s,i,p)$, is computed using Equation (8), where $|MK(s,i)|$, $|NK(s)|$ and $|NK(i)|$ means the number of items in the corresponding sets.

$$U_m(s,i,p) = \frac{|MK(s,i)|}{|NK(s)|} * \frac{|MK(s,i)|}{|NK(i)|} \quad (8)$$

2) Load Reduced

FDCC is designed to reduce the load to message brokers. To determine the load reduction through using FDCC, the input bandwidth utilisation, output bandwidth utilisation, and matching utilisation are measured for the cases with and without applying FDCC, respectively.

3) Communication Overhead

In addition to the communication overhead introduced by load detection described in Section 4.5.3, FDCC introduces an extra communication overhead. This is due to the message exchanges as part of the feedback from each subscriber. This is represented as the number of feedback messages received per second over the matching capacity of the broker.

5.3.4 Validation Results

Similar to the experiment for DRD-LB, the experiments to evaluate the performance of FDCC in the limited resource case were repeated 500 times. For each experiment, the experiment setup, including the broker capacity, the message dissemination rate, message size, number of publishers and subscribers vary randomly within a given range, e.g., for the matching capacity of a broker, this ranges from 20000 msg/s to 30000 msg/s, the same value adopted for the DRD-LB experiments (see Section 4.5.1). To evaluate the hypotheses proposed in Section 5.3.2, the performance measurements specified in Section 5.3.3 are measured in each experiment.

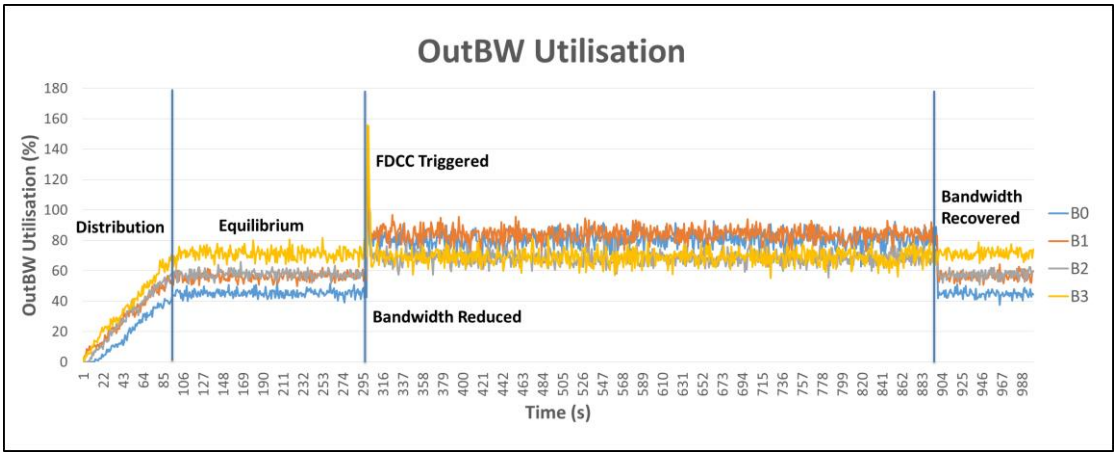

Figure 5-3 OutBW Utilisation with FDCC

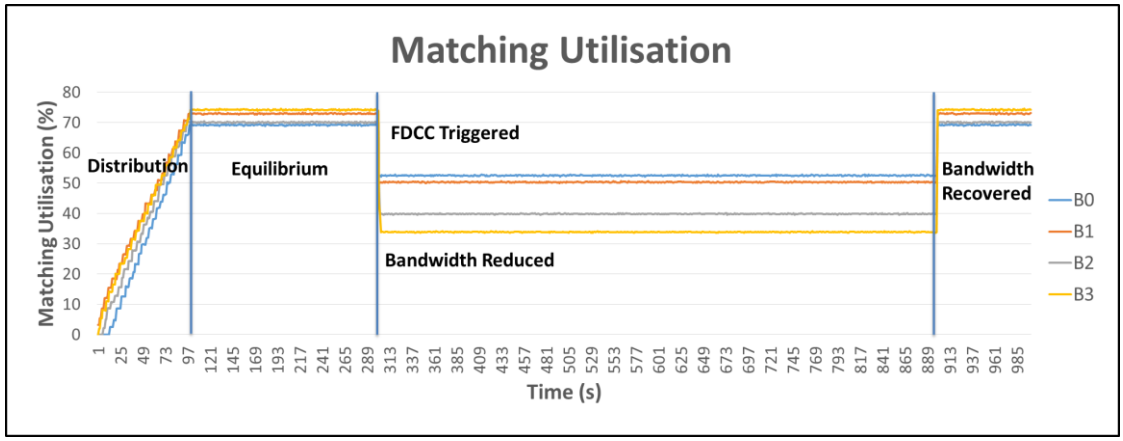

Figure 5-4 Matching Utilisation with FDCC

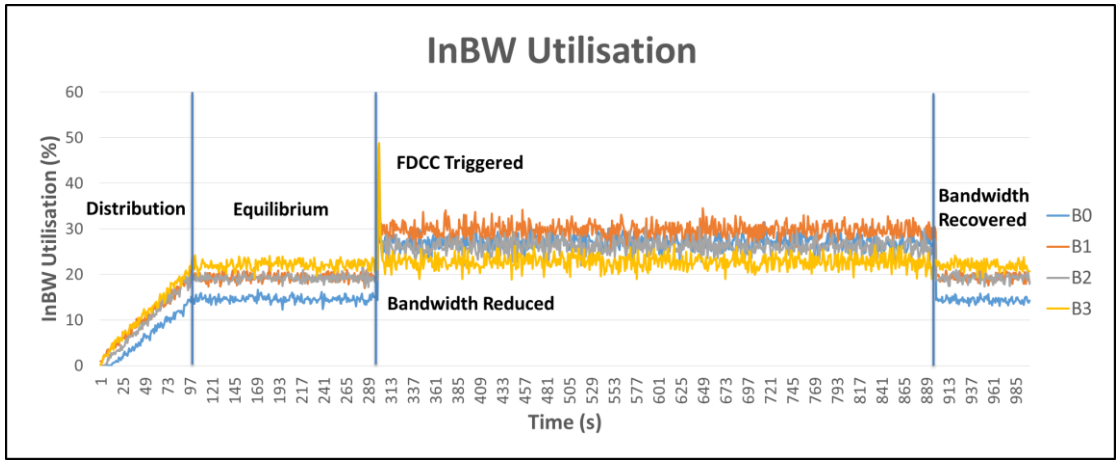

Figure 5-5 InBW Utilisation with FDCC

Figure 5-3, Figure 5-4 and Figure 5-5 show results of outBW utilisation, matching utilisation, and inBW utilisation measured in one experiment. Following a bandwidth reduction, all the brokers become overloaded, and therefore the conventional load balancing method can no longer work effectively. FDCC is then triggered to limit the rate of some overactive publishers to reduce the load to brokers. In this experiment, each subscriber sends a feedback message every second to compute the utility of the publisher, the utility of the topic and the quality of the experience for the subscribers. The total feedback message exchange rate equals to the number of subscribers. Therefore, the communication overhead caused by feedback message exchange can be obtained as the ratio of *the number of subscribers* to *the capacity of the broker*. For example, in this experiment, the communication overhead for feedback message exchange of broker B0 is 435/26015. The load reduction is also measured. In this experiment, the average load reduction for outBW utilisation, inBW utilisation, and

matching utilisation is 17.5%, 13.3% and 16.2% respectively. For the total 500 experiments, the average load reduced for the three load metrics are 19.2%, 15.7% and 16.1% respectively.

To prove Hypothesis 2, a null hypothesis H_{null2} and its alternative hypothesis H_{alt2} are introduced. H_{null2} assumes that no matter if FDCC is adopted or not, the load on brokers is similar; while H_{alt2} assumes that FDCC reduces load to brokers. In this case, the outBW utilisation, measured before and after applying FDCC, is retrieved for each experiment to form two samples. S_{out_before} and S_{out_after}. The data in each sample are indexed and paired. Similar to the Wilcoxon signed-rank test designed to prove Hypothesis 1 (Section 4.5.4), S_{out_before} and S_{out_after} are taken as the input parameters into a Wilcoxon signed-rank test, achieved using R's "wilcox.test()" function. The p value is then computed to be p<2.2e-16, which is close to zero. This means that H_{null2} rarely happens and $H_{alt}2$ is shown to be true, i.e., FDCC reduces load to brokers.

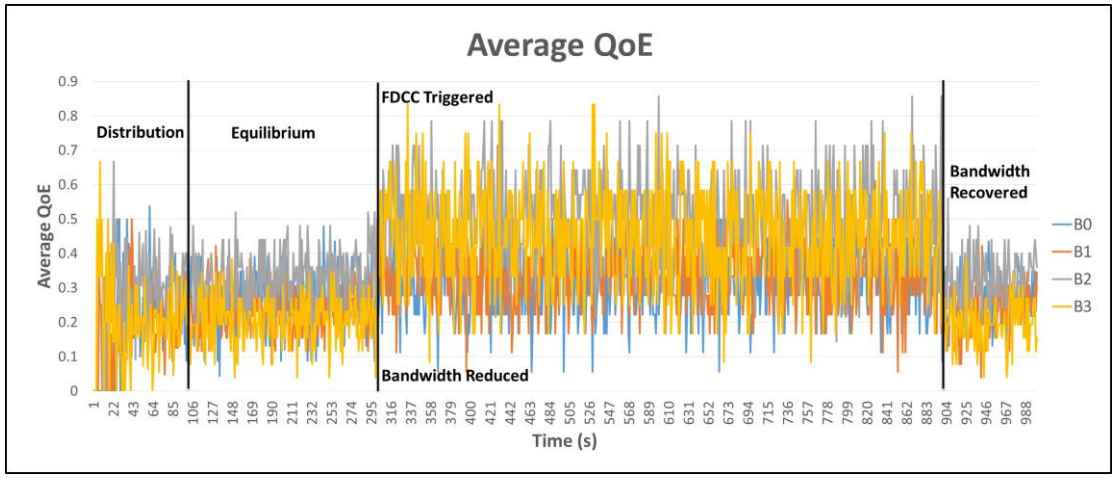

Figure 5-6 Average QoE with FDCC

Figure 5-6 shows the average QoE measured for one experiment with FDCC. The QoE is measured using the formula specified in Section 5.3.3. As shown in Figure 5-6, the average QoE varies significantly during the load distribution phase. This is because any new client that starts a new message dissemination process may influence the QoE, e.g, QoE increases if the new publishers publish messages that are more important, or QoE decreases if less important messages are published by a new publisher. In the equilibrium phase, the QoE becomes stable. After FDCC is triggered to limit the rate of overactive publishers, the average QoE for brokers is improved, as the publishing rate for messages with less important information is set with a lower value, or set to be

zero. Here, the difference between the QoE for the equilibrium phase, denoted as QoE_Before, and the QoE after FDCC triggered, denoted as QoE_After, is determined to demonstrate the improvement using FDCC, as shown in Figure 5-7.

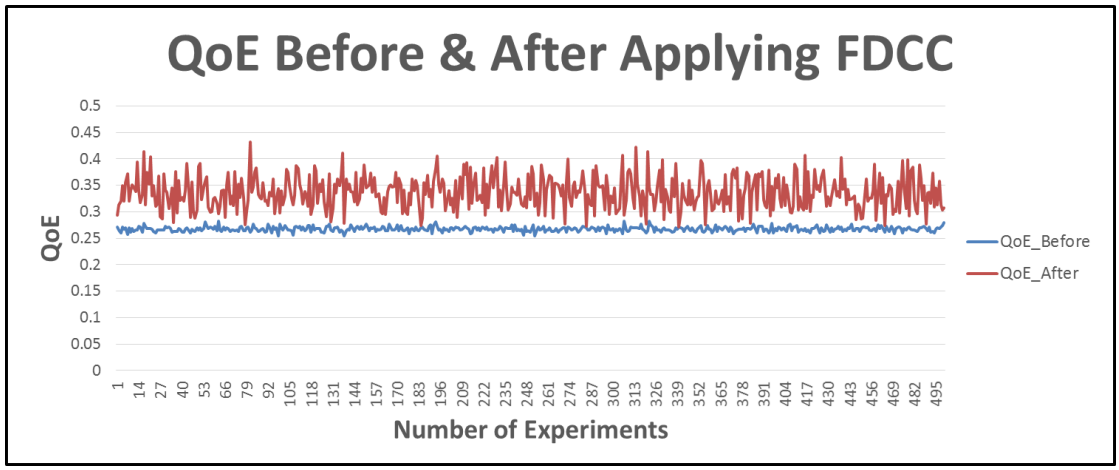

Figure 5-7 QoE Before & After Applying FDCC for the 500 Experiments

The QoE Improved, which is computed as (QoE_After – QoE_Before) / Qoe_Before is also measured, as shown in Figure 5-8. The results show that the QoE is improved after applying FDCC.

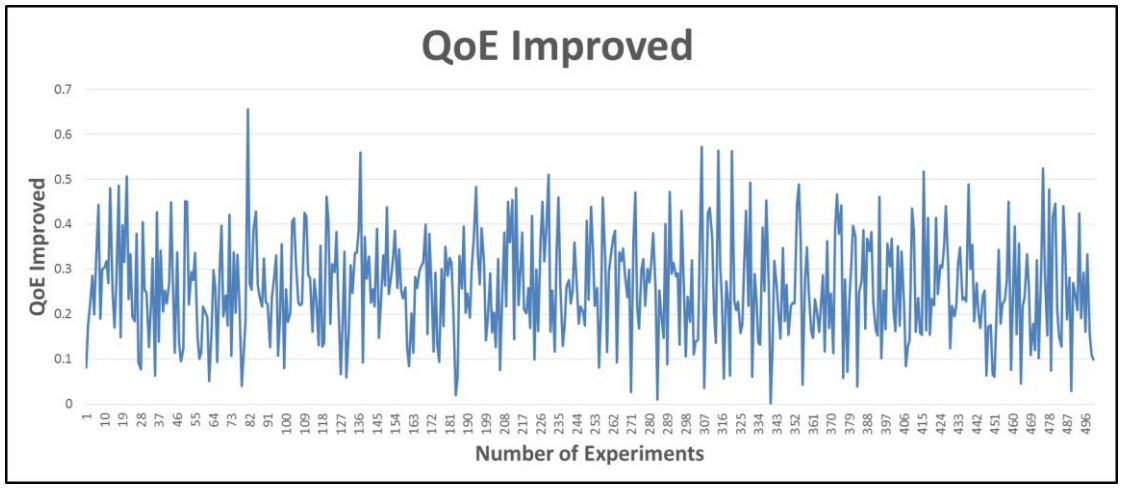

Figure 5-8 QoE Improvement Using FDCC

The overall average QoE before and after FDCC is also measured, and the values are 0.2671 and 0.3348 respectively. The QoE is improved by 25.4% in average, as shown in Figure 5-9.

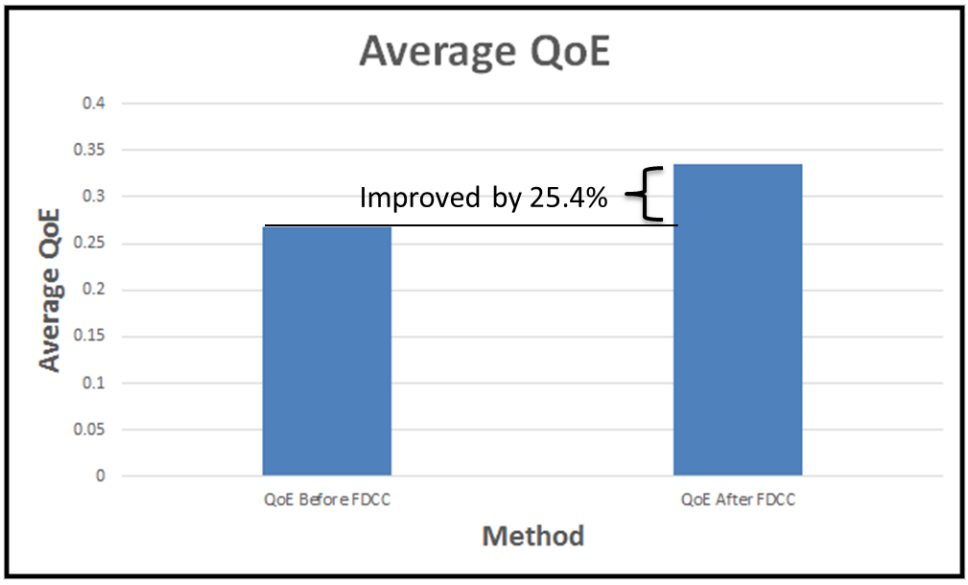

Figure 5-9 Comparison between Average QoE for 500 Experiments Before & After Applying FDCC

To prove Hypothesis 3, a null hypothesis H_{null3} and an alternative hypothesis H_{alt3} are introduced. H_{null3} assumes that the QoE after applying FDCC is similar to that before applying FDCC, while H_{alt3} assumes that the QoE after applying FDCC is greater than that before applying FDCC. In this case, the QoE that is measured before and after applying FDCC for each experiment is retrieved to form two samples, $S_{QoE\text{-}before}$ and $S_{QoE\text{-}after}$. The data in each sample are indexed and paired. Similar to the Wilcoxon signed-rank test designed to prove Hypothesis 1 (Section 4.5.4) and Hypothesis 2, $S_{QoE\text{-}before}$ and $S_{QoE\text{-}after}$ are taken as the input parameters into a Wilcoxon signed-rank test, achieved using R's "wilcox.test()" function. The p value is then computed to be $p<2.2e\text{-}16$, which is close to 0. This means that H_{null3} rarely happens and H_{alt3} is then proved, i.e., the QoE after applying FDCC is greater than that before applying FDCC. Hence, Hypothesis 3 is true.

5.4 Summary

In this chapter, a feedback driven congestion control method, named FDCC is described, which is proposed to supplement the load balancing method DRD-LB to manage the load of the H-E broker overlay for the limited resource case. FDCC limits the publishing rate of overactive publishers, thus reducing the load to brokers and improve the average QoE for subscribers. Five hundred simulation-based experiments were carried out to

evaluate the performance of FDCC. The results show that FDCC is able to reduce load to brokers by 15% and improve the QoE by 25% on average.

6 CONCLUSION & FUTURE WORK

6.1 Conclusions

This thesis selects a TWS as the target application domain of interest. Here, PSMOMs are used for the dissemination of sensor data, warning messages and evacuation signals. When a crisis happens, e.g., earthquake, the physical or underlying network may be damaged, which can result in broken network links or a decrease in network bandwidth capacity. In addition, more messages are generated and published by several sources: increased sampling rate of sensors to monitor the physical environment; increased message exchange by human beings in the affected area; increased notifications and alerts from authorities. Therefore, brokers that serve the publishers and subscribers may become overloaded and suffer performance degradation.

Existing load management solutions need to be improved when they are applied to TWS, as they tend to focus on the surplus resource case but ignore the case that there are not enough system resources to support offloading, in which the load to the broker needs to be reduced. In addition, existing solutions do not consider the delay of time-sensitive subscriptions, which needs to be minimised to ensure important messages to be delivered in time. To overcome these limitations, based upon the PEER framework, an enhanced load management framework, named ePEER is proposed to provide load management for TWS. In ePEER, brokers are organised in an H-E broker overlay, which satisfies the communication requirements in TWS and reduces the load to head brokers. Distributed MAs are adopted to construct and maintain the broker overlay and support load management. A delay-sensitive load balancing method, named DRD-LB, and a feedback-driven congestion control model, named FDCC are introduced to manage the load of PSMOM for the surplus resource and the limited resource case respectively. FDCC is considered as a supplementary method to DRD-LB when the system resource limits are reached and cannot be scaled up for a certain time. Based upon simulation-based experiments, DRD-LB is shown to outperform the state of the art load balancing method proposed for PEER framework and is accompanied by far less delay-influence on the TWS. In addition, FDCC also demonstrates its ability to reduce the load to brokers and maintain the QoE for subscribers in the limited resource case.

6.2 Current Limitations and Future Work

6.2.1 Limitations of the Current Approach

The proposed load management framework has some limitations that can be improved as follows.

- When a load management framework is deployed in a large-scale federated overlay, e.g., with hundreds of clusters, as there is no cluster-level load distribution technique, matched publishers and subscribers may be deployed in many different clusters and therefore an additional network load for message dissemination needs to be introduced.
- In this design, with respect to [7, 21], it is assumed that every component works in an ideal situation, e.g., all the subscribers have enough bandwidth capacity to receive messages and provide an instantaneous ranking feedback for each received message. However, in practice, subscribers that subscribe to particular topics may have a low bandwidth capacity to receive all the published messages with the same topic. This is referred to as the "slow subscriber" problem, which builds the message queue up.
- The proposed work targets managing the load for federated brokers in PSMOM, but does not consider the integration with other platforms, e.g., use of a Cloud, to manage the load by adding and removing brokers on demand.

6.2.2 Future Work

According to the research objectives, the current limitations, and requirements of a TWS system, the following work is proposed as future work.

- A cluster-level load distribution method can be investigated, which aims to reduce the network load by aggregating clusters that are located near to each other to form an aggregated cluster that serves clients with similar interests. Matched publishers and subscribers are assigned to the same cluster when they register with the system.
- A more practical utility ranking process can be developed, which measures the utility of each message based upon the semantic relationships between the interests of subscribers and the message content.

- An enhanced time control mechanism can be investigated, which is designed to manage slow subscribers by periodically dropping publications with less important information from the corresponding message queues to reduce the load of brokers.
- The integration with a Cloud computing cluster can be investigated to improve the elasticity of PSMOM, e.g., the creation and removal of brokers on demand. A broker-creation on demand facility helps maintain the system performance, e.g., throughput and delay, when further system capacity is required for the case when some nodes that host brokers fail. In practice, a system node, e.g., a virtual machine that hosts broker(s) can crash due to a failure of the operating system or due to a hardware failure. The time costs to fix a node crash problem may last from minutes to days and during this time, brokers running on the failure node do not work. This may cause other brokers to become overloaded, as the load on brokers running on a failed node needs to be distributed to other brokers. In this case, if a PSMOM is integrated as part of a Cloud computing system, new brokers running in another Cloud node can be automatically restarted to replace the failed ones to help maintain the performance of message exchange services. The broker-removal on demand facility saves system resources when fewer brokers are required to provide message exchange services.
- Regarding the resilience requirements of a TWS, fault tolerance techniques such as topic mirroring and geo-resilient routing, can be investigated to extend the load management framework. These techniques can be used to improve further the resilience of the PSMOM system to enable messages to be received by matched subscribers even if brokers or links to brokers fail.
- Regarding the requirements of using a mobile device to capture the live information within an environment crisis area to improve the ground truth information, a QoS adapted multimedia messaging service can be investigated. This messaging service tends to enable human beings within the disaster area to publish messages containing live video and audio information to the warning centre using mobile phones. The quality of the video and audio should be adapted to the available bandwidth capacity. In addition, content caching and delayed publication could be supported for the situation when a network segment or link becomes unavailable at specific times but recovers after a while.

REFERENCES

1. Wiltshire, A., *Developing Early Warning Systems: A Checklist*, in *Third International Conference on Early Warning*. 2006: Bonn, Germany. p. 27-29

2. Waidyanatha, N., *Towards a Typology of Integrated Functional Early Warning Systems.* International Journal of Critical Infrastructures, 2010. **6**(1): p. 31-51.

3. Intergovernmental Oceanographic Commission, *Interim Operational Users Guide for the Tsunami Early Warning and Mitigation System in the North-eastern Atlantic, the Mediterranean and Connected Seas (NEAMTWS)*. 2011.

4. Intergovernmental Oceanographic Commission. *NEAMTWS - Structural Elements of the TWS*. Available from: http://www.ioc-tsunami.org/index.php?option=com_content&view=article&id=17&Itemid=17&lang=en.

5. Braddock, R., *Sensitivity analysis of the tsunami warning potential.* Reliability Engineering & System Safety, 2003. **79**: p. 225-228.

6. Jin, D. and Lin, J., *Managing Tsunamis through Early Warning Systems : a Multidisciplinary Approach.* Ocean & Coastal Management, 2011. **54**: p. 189-199.

7. Cheung, A.K.Y. and Jacobsen, H.-A., *Load Balancing Content-Based Publish/Subscribe Systems.* ACM Transactions on Computer Systems, 2010. **28**(4): p. 1-55.

8. Pietzuch, P.R. and Bhola, S. *Congestion control in a reliable scalable message-oriented middleware*. in *Proc. of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. 2003. Rio de Janeiro, Brazil: p. 202-221.

9. Wang, J., Jiang, P., Bigham, J., Chew, B., Novkovic, M., and Dattani, I. *Adding resilience to message oriented middleware*. in *Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems*. 2010. London, United Kingdom: p. 89-94.

10. Wang, J., Bigham, J., and Murciano, B. *Towards a Resilient Message Oriented Middleware for Mission Critical Applications*. in *ADAPTIVE 2010, The Second*

International Conference on Adaptive and Self-Adaptive Systems and Applications. 2010. Lisbon, Portugal: p. 46-51.

11. Bigham, J. and Novkovic, M. *Adding Resilience to Message Oriented Middleware: The GEMOM Approach*. in *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*. 2010. Genoa, Italy: p. 292-293.

12. Tock, Y., Naaman, N., Harpas, A., and Gershinsky, G. *Hierarchical Clustering of Message Flows in a Multicast Data Dissemination System*. in *Proceedings of International Conference Parallel and Distributed Computing and Systems*. 2005. Phoenix, AZ, USA: p. 320-326.

13. Schuler, C., Schuldt, H., and Schek, H.-J., *Supporting Reliable Transactional Business Processes by Publish/Subscribe Techniques*, in *Technologies for E-Services*. 2001, Springer Berlin Heidelberg. p. 118-131.

14. Fawcett, T. and Provost, F. *Activity monitoring: noticing interesting changes in behavior*. in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 1999. San Diego, California, USA: p. 53-62.

15. Cugola, G., Di Nitto, E., and Fuggetta, A., *The JEDI event-based infrastructure and its application to the development of the OPSS WFMS.* Software Engineering, IEEE Transactions on, 2001. **27**(9): p. 827-850.

16. Rose, I., Murty, R., Pietzuch, P., Ledlie, J., Roussopoulos, M., and Welsh, M. *Cobra: contentbased filtering and aggregation of blogs and RSS feeds*. in *Proc. of the 4th USENIX conference on Networked systems design & implementation*. 2007. Cambridge, MA: p. 3-3.

17. Liu, H., Ramasubramanian, V., and Sirer, E.G. *Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews*. in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. 2005. Berkeley, CA: p. 3-3.

18. Abie, H., Dattani, I., Novkovic, M., Bigham, J., Topham, S., and Savola, R. *GEMOM - Significant and Measurable Progress beyond the State of the Art*. in

Systems and Networks Communications, 2008. ICSNC '08. 3rd International Conference on. 2008. Sliema: p. 191-196.

19. Abie, H., Savola, R.M., and Dattani, I. *Robust, Secure, Self-Adaptive and Resilient Messaging Middleware for Business Critical Systems*. in *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World:*. 2009. Athens: p. 153-160.

20. Wang, J., Bigham, J., and Wu, J. *Enhance Resilience and QoS Awareness in Message Oriented Middleware for Mission Critical Applications*. in *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*. 2011. Las Vegas, USA: p. 677-682.

21. Cheung, A.K.Y. and Jacobsen, H.-A., *Dynamic load balancing in distributed content-based publish/subscribe*, in *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*. 2006: Melbourne, Australia. p. 141-161.

22. Collins, L. *Redundancy lesson for Japan's comms infrastructure*. 2011; Available from: http://eandt.theiet.org/magazine/2011/04/japans-infrastructure.cfm.

23. An, X.-F. and Bian, L.-Y. *Design of Message-Oriented Middleware of Distance Teaching Platform Based on Distributed Message Control*. in *Computational Aspects of Social Networks (CASoN), 2010 International Conference on*. 2010. Tiyuan, China: p. 141-143.

24. Lewis, D., Keeney, J., O'Sullivan, D., and Guo, S., *Towards a Managed Extensible Control Plane for Knowledge-Based Networking*, in *Large Scale Management of Distributed Systems*. 2006, Springer Berlin Heidelberg. p. 98-111.

25. Li, H. and Jiang, G. *Semantic message oriented middleware for publish/subscribe networks*. in *SPIE - The International Society for Optical Engineering*. 2004. USA: p. 124-133.

26. Parkin, S., Ingham, D., and Morgan, G., *A message oriented middleware solution enabling non-repudiation evidence generation for reliable web*

services, in *4th International Service Availability Symposium, ISAS* 2007: Durham, NH, USA. p. 98-111.

27. Kim, M., Karenos, K., Ye, F., Reason, J., Lei, H., and Shagin, K. *Efficacy of techniques for responsiveness in a wide-area publish/subscribe system*. in *11th International Middleware Conference Industrial track*. 2010. Bengaluru, India.

28. Baldoni, R., Contenti, M., and Virgillito, A., *The evolution of publish/subscribe communication systems*, in *Future directions in distributed computing*. 2003, Springer-Verlag. p. 137-141.

29. Venkatesh, V., Morris, M.G., Davis, G.B., and Davis, F.D., *User acceptance of information technology: Toward a unified view.* MIS Quarterly, 2003. **27**(3): p. 425-478.

30. Corporation, O. *Java Message Service API Rev. 1.1*. 2002; Available from: http://java.sun.com/products/jms/.

31. Terry, S. and Shawn, T., *Enterprise JMS programming*. 2002: John Wiley & Sons, Inc.

32. Hapner, M., Burridge, R., Sharma, R., Fialli, J., and Haase, K., *Java Message Service API Tutorial and Reference: Messaging for the J2EE Platform*. 2002: Addison-Wesley.

33. Fiorano Software, I. *FioranoMQTM: Meeting the Needs of Technology and Business*. Available from: http://www.fiorano.com/whitepapers/

34. Tibco Software, I. *TIBCO Enterprise Message Service*. 2004; Available from: http://www.tibco.com/products/automation/enterprise-messaging/enterprise-message-service.

35. TIBCO Software Inc. *TIBCO Rendezvous Messaging Middleware*. Available from: http://www.tibco.com/products/soa/messaging/rendezvous/.

36. IBM Corporation. *IBM WebSphere MQ 6.0*. 2005; Available from: http://www-01.ibm.com/software/integration/wmq/.

37. *IBM WebSphere MQ Homepage*. Available from: http://www-306.ibm.com/software/integration/wmq/.

38. *IBM MQ Fundamentals*. Available from: http://www.redbooks.ibm.com/abstracts/sg247128.html.

39. *IBM WebSphere MQ Information Centre*. Available from: http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp.

40. Rabbit Technologies Ltd. *RabbitMQ – Messaging that just works*. Available from: http://www.rabbitmq.com/.

41. AMQP Working Group. *AMQP: Advanced Message Queuing Protocol*. Available from: http://www.amqp.org/.

42. Vinoski, S., *Advanced Message Queuing Protocol.* Internet Computing, IEEE, 2006. **10**(6): p. 87-89.

43. Kramer, J., *Advanced message queuing protocol (AMQP).* Linux J., 2009. **2009**(187): p. 3.

44. *MQTT Version 3.1.1*. Available from: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

45. *The Simple Text Oriented Messaging Protocol*. Available from: http://stomp.github.io.

46. Chockler, G., Melamed, R., Tock, Y., and Vitenberg, R. *Constructing Scalable Overlays for Pub-Sub with Many Topics Problems, Algorithms, and Evaluation*. in *the 26th Annual ACM Symposium on Principles of Distributed Computing*. 2007. Portland: p. 109-118.

47. Chen, C., Jacobsen, H.A., and Vitenberg, R. *Divide and Conquer Algorithms for Publish/Subscribe Overlay Design*. in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. 2010. Genoa, Italy: p. 622-633.

48. Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R.E., and Sturman, D.C. *An efficient multicast protocol for content-based publish-subscribe systems*. in *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*. 1999. Austin, TX: p. 262-272.

49. Carzaniga, A., Rosenblum, D.S., and Wolf, A.L., *Design and evaluation of a wide-area event notification service.* ACM Trans. Comput. Syst., 2001. **19**(3): p. 332-383.

50. Fabret, F., Jacobsen, H.A., Llirbat, F., Pereira, J., Ross, K.A., and Shasha, D., *Filtering algorithms and implementation for very fast publish/subscribe systems*, in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*. 2001: Santa Barbara, California, USA. p. 115-126.

51. Li, G., Hou, S., and Jacobsen, H.A. *Routing of XML and XPath Queries in Data Dissemination Networks*. in *Distributed Computing Systems, 2008. ICDCS '08. The 28th International Conference on*. 2008. Beijing, China: p. 627-638.

52. Pietzuch, P.R. and Bacon, J.M. *Hermes: a distributed event-based middleware architecture*. in *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*. 2002. Viena, Austria: p. 611-618.

53. Li, G. and Jacobsen, H.A., *Composite subscriptions in content-based publish/subscribe systems*, in *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*. 2005: Grenoble, France. p. 249-269.

54. Eugster, P., Felber, P., Guerraoui, R., and Kermarrec, A.M., *The many faces of publish/subscribe.* ACM Computing Surveys, 2003. **35**(2): p. 114-131.

55. Baldoni, R., Beraldi, R., Querzoni, L., and Virgillito, A., *Efficient Publish/Subscribe Through a Self-Organizing Broker Overlay and its Application to SIENA.* Comput. J., 2007. **50**(4): p. 444-459.

56. Pietzuch, P.R., *Hermes: A scalable event-based middleware.* 2004. **Doctor of Philosophy**: p. 180.

57. Chen, Y. and Schwan, K. *Opportunistic overlays: efficient content delivery in mobile ad hoc networks*. in *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*. 2005. Grenoble, France: p. 354-374.

58. Chew, B. and Bigham, J. *Bottleneck detection and forecasting in Message-Oriented-Middleware*. in *European Safety and Reliabitily Conference*. 2011. Troyes, France.

59. Tran, P., Greenfield, P., and Gorton, I. *Behavior and performance of message-oriented middleware systems*. in *Distributed Computing Systems Workshops,*

2002. Proceedings. 22nd International Conference on. 2002. Viena, Austria: p. 645-650.

60. Henjes, R. *Performance Evaluation of Publish/Subscribe Middleware Architectures*. 2010; Available from: http://www3.informatik.uni-wuerzburg.de/diss/diss_28.pdf.

61. Remzi, H.A.-D. and A.-D., A.C. *Operating Systems: Three Easy Pieces* Available from: http://pages.cs.wisc.edu/~remzi/OSTEP/.

62. Thingom, C. and Suma, V. *Ensured availability of resources in a highly reliable mode through enhanced approaches for effective disaster management in cloud*. in *Electronics and Communication Systems (ICECS), 2014 International Conference on*. 2014. Marseille, France: p. 1-6.

63. Casalicchio, E. and Morabito, F. *Distributed subscriptions clustering with limited knowledge sharing for content-based publish/subscribe systems*. in *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*. 2007. Cambridge, MA, USA: p. 105-112.

64. Riabov, A., Zhen, L., Wolf, J.L., Yu, P.S., and Li, Z. *Clustering algorithms for content-based publication-subscription systems*. in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*. 2002. Viena, Austria: p. 133-142.

65. Riabov, A., Zhen, L., Wolf, J.L., Yu, P.S., and Li, Z. *New algorithms for content-based publication-subscription systems*. in *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*. 2003. Viena, Austria: p. 678-686.

66. Wong, T., Katz, R., and McCanne, S. *An evaluation of preference clustering in large-scale multicast applications*. in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. 2000. Tel Aviv: p. 451-460 vol.2.

67. Amol, D. and Rajesh, P. *A Review on Active Queue Management Techniques of Congestion Control*. in *Electronic Systems, Signal Processing and Computing Technologies (ICESC), 2014 International Conference on*. 2014. Marseille, France: p. 166-169.

68. Kafi, M., Djenouri, D., Ben-Othman, J., and Badache, N., *Congestion Control Protocols in Wireless Sensor Networks: A Survey.* Communications Surveys & Tutorials, IEEE, 2014. **PP**(99): p. 1-22.

69. Jenolin Flora, D.F., Kavitha, V., and Muthuselvi, M. *A survey on congestion control techniques in Wireless Sensor Networks*. in *Emerging Trends in Electrical and Computer Technology (ICETECT), 2011 International Conference on*. 2011. India: p. 1146-1149.

70. Escudero-Sahuquillo, J., Gran, E.G., Garcia-Garcia, P.J., Flich, J., Skeie, T., Lysne, O., Quiles, F.J., and Duato, J., *Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks.* Parallel and Distributed Systems, IEEE Transactions on, 2014. **PP**(99): p. 1-1.

71. Soelistijanto, B. and Howarth, M.P., *Transfer Reliability and Congestion Control Strategies in Opportunistic Networks: A Survey.* Communications Surveys & Tutorials, IEEE, 2014. **16**(1): p. 538-555.

72. Shiang, H.-P. and van der Schaar, M., *A Quality-Centric TCP-Friendly Congestion Control for Multimedia Transmission.* Multimedia, IEEE Transactions on, 2012. **14**(3): p. 896-909.

73. Wang, C., Li, B., Sohraby, K., Daneshmand, M., and Hu, Y., *Upstream congestion control in wireless sensor networks through cross-layer optimization.* Selected Areas in Communications, IEEE Journal on, 2007. **25**(4): p. 786-795.

74. Kang, J., Zhang, Y., and Nath, B., *TARA: Topology-Aware Resource Adaptation to Alleviate Congestion in Sensor Networks.* Parallel and Distributed Systems, IEEE Transactions on, 2007. **18**(7): p. 919-931.

75. Zawodniok, M. and Jagannathan, S., *Predictive Congestion Control Protocol for Wireless Sensor Networks.* Wireless Communications, IEEE Transactions on, 2007. **6**(11): p. 3955-3963.

76. Shi, K., Shu, Y., Yang, O., and Luo, J., *Receiver-Assisted Congestion Control to Achieve High Throughput in Lossy Wireless Networks.* Nuclear Science, IEEE Transactions on, 2010. **57**(2): p. 491-496.

77. Jerzak, Z. and Fetzer, C. *Handling Overload in Publish/Subscribe Systems*. in *Distributed Computing Systems Workshops, 2006. ICDCS Workshops 2006. 26th IEEE International Conference on*. 2006. Lisboa, Portugal: p. 32-32.

78. Afrianfar, S., *Optimizing Publish/Subscribe Systems with Congestion Handling.* Department of Computer Science and Engineering, 2008. **Master of Science**: p. 72.

79. Abie, H., Savola, R.M., Bigham, J., Dattani, I., Rotondi, D., and Bormida, G.D., *Self-Healing and Secure Adaptive Messaging Middleware for Business-Critical Systems.* International Journal on Advances in Security, 2010. **3**(1 & 2): p. 34-51.

80. Jia, Y., Bodanese, E., and Bigham, J., *Model Checking of the Reliability of Publish/Subscribe Structure Based System*, in *IEEE International Conference on Communications in China*. 2012: Beijing, China. p. 173-178.

81. Jia, Y., Bodanese, E., and Bigham, J., *Checking the Robustness of a Publish/Subscribe Based Message Oriented System*, in *International Congress on Ultra Modern Telecommunications and Control Systems*. 2012: St. Petersburg, Russia. p. 280 - 285.

82. Jia, Y., Bodanese, E., Phillips, C., Bigham, J., and Tao, R., *Improved Reliability of Large Scale Publish/Subscribe based MOMs using Model Checking*, in *IEEE/IFIP Network Operations and Management Symposium*. 2014: Krakow, Poland. p. 1 - 8.

83. Wang, X., Tokarchuk, L., Cuadrado, F., and Poslad, S. *Exploiting hashtags for adaptive microblog crawling*. in *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*. 2013. Niagara Falls, Canada: p. 311-315.

84. TRIDEC. *TRIDEC: Collaborative, Complex and Critical Decision Support in Evolving Crises*. Available from: http://www.tridec-online.eu/.

APPENDIX A: AUTHOR'S CONTRIBUTION

- **Tao**, R., Poslad, S., MoBgraber, J., Middleton, S.E., and Hammitzsch, M. *Scalable and Resilient Middleware to Handle Information Exchange during Environment Crisis*, in European Geosciences Union General Assembly (EGU 2012), Austria, 2012.
- **Tao, R.,** and Poslad, S. *Delay Sensitive Distributed Sensor Data Exchange for an IoT*, in Proc. of Int. Workshop on Adaptive Security (ASPI '13), Switzerland, 2013.
- **Tao**, **R.**, Poslad, S., and Bigham, J. *Resilient Delay Sensitive Load Management in Environment Crisis Messaging System*, in Proc. of the Eighth Int. Conf. on System and Networks Communications (ICSNC'13), Italy, 2013.
- Tavakoli, S., Poslad, S., Dattani, I., **Tao, R.,** and Haner, R. *A System Infrastructure to Handle Large Data Stream Exchange and Collaboration during Evolving Environment Crises*, in 18th Annual Conference of Int. Emergency Management Society (TIEMS'11), Romania, 2011.
- MoBgraber, J., Middleton, S.E., and **Tao, R.** *A Geo-Distributed System Architecture for Different Domains*, in European Geosciences Union General Assembly (EGU 2013), Austria, 2013.
- MoBgraber, J., Chaves, F., Middleton, S.E., Zlatev, Z., and **Tao**, **R.** *The Seven Main Challenges of an Early Warning System Architecture*, in Proc. Of the 10th Int. ISCRAM Conf., Germany, 2013.
- Jia, Y., Bodanese, E., Phillips, C., Bigham, J., and **Tao**, **R.** *Improved Reliability of Large Scale Publish/Subscribe based MOMs using Model Checking*, in Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS'14), Poland, 2014.
- Poslad, S., Middleton, S. E., Chaves, F., **Tao, R**., Necmioglu, O., Bügel, U. A Semantic IoT Early Warning System for Natural Environment Crisis Management. IEEE Transaction on Emerging Topics in Computing (TETC) Special Issue on Advances in Semantic Computing, 3(2), 246 – 257, 2015

The author has also contributed to nine TRIDEC WP3 deliverables and three project review demos.

APPENDIX B: RELATION OF THIS PHD TO TRIDEC PROJECT

This PhD was co-funded[8] by the EU FP7 project TRIDEC that focused on new approaches and technologies for intelligent geo-information management in complex and critical decision making processes in Earth sciences [84]. In TRIDEC, QMUL was the leader for Work Package 3 that focused on Resilience, Performance and Scalability Modelling, architectural and component design, implementation and integration. This PhD contributed the core resilient and scalable messaging middleware model based upon a PSMOM design for use in the TRIDEC System to support two applications, Tsunami and subsurface drilling for oil and geothermal energy. The research and development of this middleware is solely the result of this PhD.

The proposed PSMOM framework from the PhD, including the broker overlay, management component and message exchange interface, has been developed and deployed to provide message exchange services for two real EWS applications developed in project TRIDEC for two scenarios:

- The first scenario concerns a large group of experts working collaboratively in crisis centres and government agencies monitoring the physical environment using networked sensors. The goal is to make critical decisions and to save lives as well as infrastructural and industrial facilities in evolving tsunami crises.
- The second scenario concerns a large group of consulting engineers and financial analysts from energy companies working collaboratively in sub-surface drilling operations. Their common objective is to monitor drilling operations in real-time using sensor networks, optimising drilling processes and critically detecting unusual trends of drilling systems functions. This prevents operational delays, financial losses, and environmental accidents.

In both scenarios, the broker overlay consists of three broker clusters, located in different geo-locations and each cluster following the head-edge cluster design

[8] It is also funded by a QMUL PhD Scholarship

described in Section 4.2. Fig B.1 shows generic system architecture for the two TRIDEC applications with the proposed PSMOM framework.

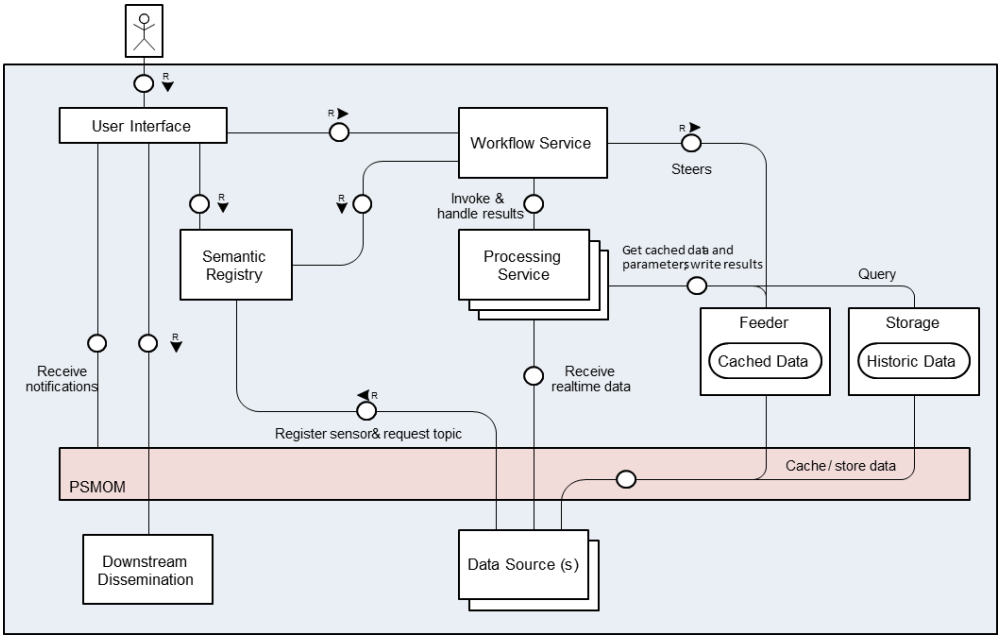

Fig. B.1 Generic TRIDEC Architecture

To give a better understanding of how the proposed framework is used to enable communication for the system of system architecture in TRIDEC, Fig. B.2 shows an example of using a broker overlay to enable communication in a Tsunami early warning system.

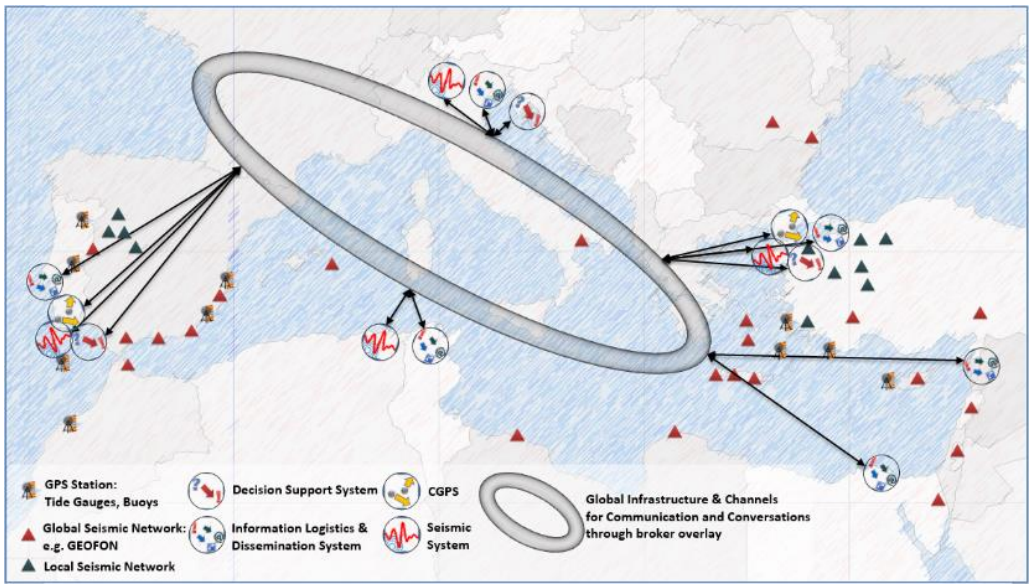

Fig. B.2 Communication in a TWS through Broker Overlay

Both applications have been demonstrated with live demos in the TRIDEC project third year review meeting in Lisbon, 28th Nov. 2013. Fig B.3 and Fig. B.4 show the user interface for live demos for drilling and Tsunami scenarios respectively. In both demos, all the data exchanged are through the proposed PSMOM framework.

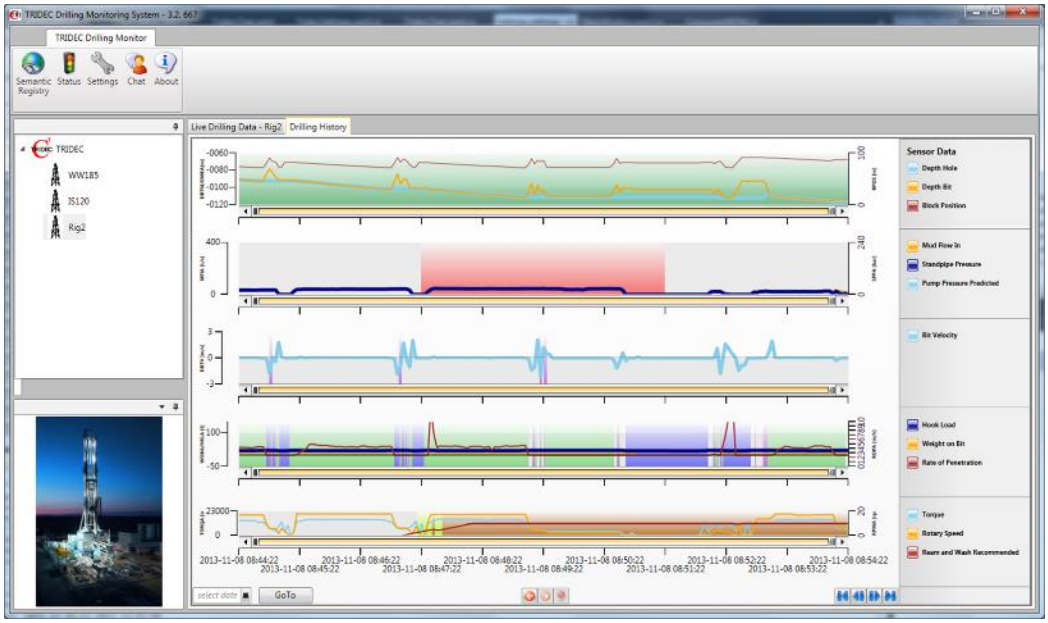

Fig. B.3 Live Demo User Interface for Drilling Scenario

In the drilling demo, live sensor data from onsite sensors are transmitted to the remote data processing centre through brokers, and the data is analysed together with the historical data. The diagram shows the drilling history of a rig named Rig2.

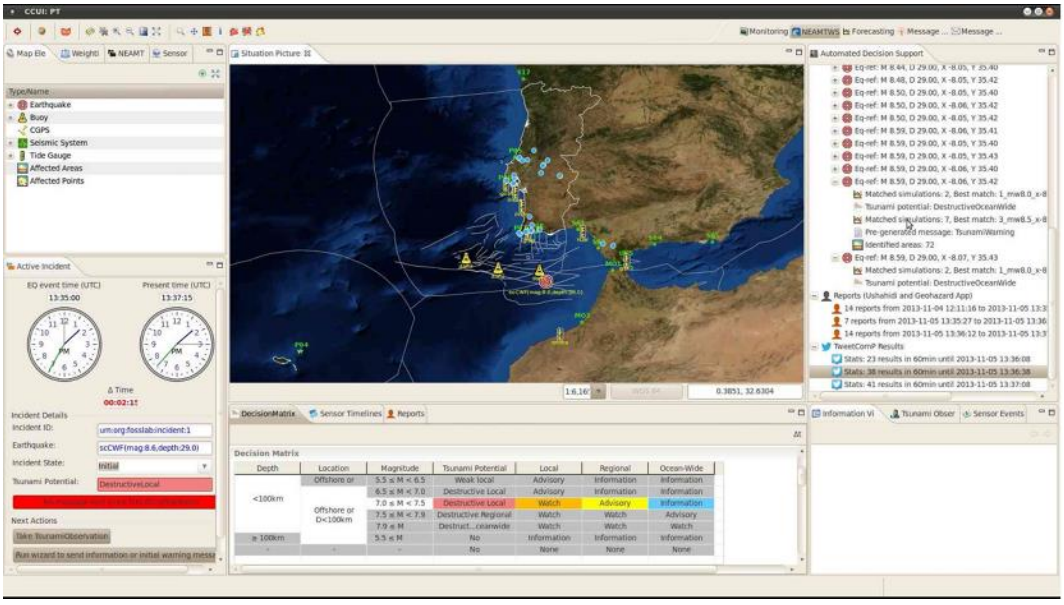

Fig. B.4 Live Demo User Interface for Tsunami Scenario

In the tsunami scenario, seismic data and sea level data is transmitted through brokers to the data processing system. The data processing system analyses the received data together with the historical data and predicts the affected areas and the height of the wave. In addition, data from social network, e.g., tweets, which is relevant to the tsunami, is gathered to analyse the behaviour of people in the affected areas. The diagram shows where the earthquake happens, the distribution of the sea level sensors and the geo-locations of the tweets relevant to the events.